［6月号別冊付録］

CASL
FORTRAN
COBOL

▽

基本文法ハンドブック

CASL
FORTRAN
COBOL

▼
基本文法ハンドブック

CASL
基本文法ハンドブック

明石ミニコン研究会
赤松　徹

▶このCASL基本文法ハンドブックは，CASLの初心者から中級者までを対象に，プログラミング上の規則や基本命令の使い方などを解説したものです。

▶初心者のかたは，最初から順に読んでください。CASLを系統的にわかりやすく解説してあります。中級者のかたは，リファレンスマニュアルとして利用してください。

▶このハンドブックは，英語の勉強にたとえれば単語帳のようなものです。本誌の連載講座でプログラミングテクニックとして解説している命令のセットが，熟語に相当します。ハンドブックと本誌の講座とを効果的に活用し，学習を進めてください。

本書の読み方

▶命令の表記

各命令の説明には，次の表記法を用います。

GR	GRの値を番号とする汎用レジスタ（ただし，0≦GR≦4）
XR	XRの値を番号とする指標レジスタ（ただし，1≦XR≦4）
SP	スタックポインタ（汎用レジスタ4番）
adr	ラベル名（ラベル名に対応する番地を示す）または10進定数（ただし，−32768≦adr≦65535とする。adrはアドレスとして0〜65535の値をもつが，32768〜65535の値を負の10進定数で記述することもできる
実効アドレス	adrとXRの内容とのアドレス加算†値またはその値が示す番地
(X)	X番地の内容。Xがレジスタの場合はレジスタの内容
[ ]	[ ]に囲まれた部分は，省略可能であることを示す XRを省略した場合は，指標レジスタによる修飾を行わない

▶命令の書き方

擬似命令，マクロ命令，機械語命令は，ラベル欄，命令コード欄，オペランド欄，注釈欄をもち，各欄は，次のとおり定義します。

ラベル欄	第1文字からラベルの文字数分（最大6文字）
命令コード欄	㋑ラベルを付けないとき――第2文字以降，任意の文字位置から ㋺ラベルを付けたとき――ラベルに続けて，少なくとも1つの空白を置いたあと，任意の文字位置から
オペランド欄	命令コードに続けて，少なくとも1つの空白を置いたあと，第72文字までとする。次の行に継続することはできない
注釈欄	行中にセミコロン（；）があると，それ以降の行の終わりまで注釈として扱う（ただし，DC命令の文字列中の「；」を除く） なお，第1文字位置に「；」がある場合，または「；」の前に空白しかない場合は，その行全体を注釈として扱う 注釈欄には，処理系で許す任意の文字を書くことができる

▶命令の形式

①R-M（レジスターメモリ）形式

ラベル	命令コード	オペランド
[label]	＊＊＊＊＊	GR, adr[, XR]

（＊＊＊＊＊は命令コード）

R-M形式の命令には，LD，ST，LEA，ADD，SUB，AND，OR，EOR，CPA，CPL，SLL，SRL，SLA，SRA命令があります。

②M（メモリ）形式

ラベル	命令コード	オペランド
[label]	＊＊＊＊＊	adr[, XR]

M形式の命令には，JPZ，JMI，JNZ，JZE，JMP，PUSH，CALL命令があります。

③その他の形式

その他の形式として次の命令があります。

ラベル	命令コード	オペランド
[label]	POP	GR
[label]	RET	空白
[label]	EXIT	空白
label	START	[実行開始番地]
空白	END	空白
[label]	DC	定数
[label]	DS	領域の語数
[label]	IN	入力領域，入力文字長
[label]	OUT	出力領域，出力文字長

† アドレス加算：被演算データを符号のない数値とみなし，その和を65536で割った剰余（和の下位16ビット）を値とする。アドレス減算もこれに準じた定義とする。

CASL
基本文法ハンドブック――目次

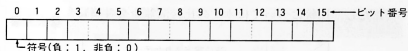
ハードウェア・COMETの仕様

①COMETは，1語16ビットの計算機であって，アクセスできるアドレスは0番地から65535番地までである。

②1語のビット構成は，次のとおりである。

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

15←ビット番号

└符号(負：1，非負：0)

③数値は，16ビットの2進数により表現する。負数は2の補数表記で表す。

④制御方式は逐次制御で，2語長の命令語をもつ。

⑤レジスタとして，GR（16ビット），PC（16ビット），FR（2ビット）をもつ。

GR（汎用レジスタ General Register）は5個あり，汎用レジスタ0番から4番までとする。この5個のレジスタは，算術演算，論理演算，比較演算，シフト演算などに用いる。このうち，1番から4番までのレジスタは，指標レジスタ(index register)としても用いる。さらに，4番のレジスタは，スタックポインタ（stack pointer）としても用いる。

スタックポインタは，スタックの最上段(stack top)のアドレスを保持しているレジスタである。

PC（プログラムカウンタ Program Counter）は，実行中の命令語の先頭アドレスを保持し，命令の実行が終わると，次に実行する命令語の先頭アドレスが設定される。一般に，命令の実行が終わるとPCに2がアドレス加算され，分岐，コール，リターン命令の場合は，新たに分岐先のアドレスが設定される。

FR（フラグレジスタ Flag Register）は，ロードアドレス命令および算術，論理，シフトの各演算命令の実行の結果，GR に設定されたデータが，負，ゼロ，正のいずれであるかについての情報，または，比較演算命令の実行により得られた，2 数間の大小関係についての情報を保持する。

すなわち，実行結果により，FR は次の表のとおりに設定される（ただし，比較については本文の比較演算命令参照）。

	GRに設定されたデータ		
	負	ゼロ	正
FRの値	10	01	00

上の表で，負は GR の符号ビットが 1，ゼロは GR の全ビットが 0，正は符号ビットが 0 でかつゼロでないデータをいう。

FR の第 1（左端の）ビットは GR の符号を示し，第 2 ビットは GR がゼロか否かを示す。

FR の値は，条件付き分岐命令で参照する。

ロードアドレス，算術，論理，シフト以外の命令の実行によって，FR の値は変更されない。

⑥命令語の構成

2 語長の命令語をもつ。その構成については定義しない。

START命令 START

▶書き方例（その他の形式）

書き方❶

ラベル	命令コード	オペランド
EX01	START	

書き方❷

ラベル	命令コード	オペランド
EX02	START	SOFT

▶命令の機能

①プログラムの先頭を定義する。

②プログラムの実行開始番地をオペランドのラベルで定義する。

③副プログラムの場合，ラベル名で入口名を定義する。

④START 命令は擬似命令で，命令語（機械語）には変換されない。

▶解説

プログラムの最初には，必ず START 命令を書かなければならない。START 命令には，ラベルを書かなければならない。

ラベルは6文字以内で，先頭の文字は英大文字でなければならない。それ以降の文字は英大文字，数字のいずれでもよい。

実行開始番地は，このプログラム内で定義されているラベル名とし，オペランドで指定したラベル名から実行が開始される。ラベル名を省略した場合は，プログラムの先頭から実行が開始される。

書き方❷の例が副プログラムとすると，ラベル EX02が副プログラムの入口名で，ラベル SOFT 番地から実行される。

なお，ほかのプログラムから副プログラム EX02を呼ぶには，

CALL　　EX02

と書く。

END命令 END

▶書き方例（その他の形式）

ラベル	命令コード	オペランド
	END	

▶命令の機能

①プログラムの終わりを定義する。

②プログラムは，START 命令に始まり，END 命令で終わる。

③ END 命令では，ラベル欄もオペランド欄も空白でなければならない。

④ END 命令は擬似命令で，命令語（機械語）には変換されない。

▶解説

START 命令と END 命令の間がプログラム単位となる。

主プログラムと副プログラムには，ともに必ず START 命令と END 命令を書かなければならない。さらに，この 2 命令は，CASL 処理系のアセンブラ部分に，原始プログラムの始まりと終わりを指示する命令であるから，命令語に変換されない。一方，EXIT 命令は，CASL 処理系のシミュレーション部分に，目的プログラムの実行終了を示す命令であるから，命令語に変換される。FORTRAN 言語の STOP 命令が EXIT 命令に対応する。

ラベル	命令コード	オペランド
MAIN	START	
	⋮	
	CALL	SUB
	⋮	
	EXIT	
	END	

主プログラム

ラベル	命令コード	オペランド
SUB	START	
	⋮	
	RET	
	END	

副プログラム

DC命令 Define Constant

▶書き方例（その他の形式）

ラベル	命令コード	オペランド
L1234	DC	1234　　　　　　；(イ)
	DC	#1234　　　　　；(ロ)
	DC	'1234'　　　　　；(ハ)
	DC	L1234　　　　　；(ニ)

▶命令の機能

定数で指定した定数データを格納する。定数には，10進定数，16進定数，文字定数，アドレス定数の4種類がある。

▶解説

例にあげた(イ)のDC命令は，10進数値1234を1語の2進数データとして格納する。ただし，オペランドで指定した10進数値が，−32768～32767の範囲にないときは，その下位16ビットを格納する。

次の(ロ)のDC命令は，4けたの16進数X〝1234″を1語の2進データとして格納する（X〝0000″≦16進数値≦X〝FFFF″）。

(ハ)のDC命令は，文字列の左端から1文字ずつ，連続する語の下位8ビットに文字データを格納する。すなわち，最初の文字'1'は第1語の第8～15ビットに，2番目の文字'2'は第2語の第8～15ビットに……と順次，文字列の文字数分，文字データを格納する。各語の第0～7ビットには0のビットが入る。文字列には，間隔および任意の図形文字（次ページの表参照）を書くことができる。ただし，文字列中にアポストロフィ（'）は書けない。アポストロフィを定義するには，次のいずれかの方法を用いる。

```
APOS    DC    #0027
        DC    39
```

さらに，文字列の長さは0（文字列が空）であってはならない。

表　JIS C6220文字コード表の一部

行＼列	02	03	04	05
0	間隔	0	@	P
1	!	1	A	Q
2	"	2	B	R
3	#	3	C	S
4	$	4	D	T
5	%	5	E	U
6	&	6	F	V
7	'	7	G	W
8	(	8	H	X
9	)	9	I	Y
10	*	:	J	Z
11	+	;	K	[
12	,	<	L	¥
13	−	=	M	]
14	.	>	N	^
15	/	?	O	

(ニ)のDC命令は，ラベル名(L1234)に対応するアドレス値を1語の2進データとして格納する。ラベルL1234番地を100番地とすると，例にあげたDC命令によって，主記憶装置の記憶内容は次のようになる。

番地	内容	
100番地	0000 0100 1101 0010	1234
101番地	0001 0010 0011 0100	#1234
102番地	0000 0000 0011 0001	'1'
103番地	0000 0000 0011 0010	'2'
104番地	0000 0000 0011 0011	'3'
105番地	0000 0000 0011 0100	'4'
106番地	0000 0000 0110 0100	L1234

DS命令 Define Storage

▶書き方例（その他の形式）

ラベル	命令コード	オペランド
A	DS	1
B	DS	3
C	DS	0

▶命令の機能

①指定した語数の領域を確保する。領域の語数は，10進定数(≧ 0)で指定する。

②領域の語数を 0 とした場合，領域は確保しない。ただし，ラベル欄のラベル名は有効である。

▶解説

いま，A 番地が300番地とすると，例にあげたDS 命令によって次のように記憶領域が割り当てられる。

ラベルA	300番地	?	?	?	?
ラベルB	301番地	?	?	?	?
	302番地	?	?	?	?
	303番地	?	?	?	?

ラベル A 番地は 1 語分確保される。ラベル B 番地は，配列 B として 3 語分確保される。ところで，ラベル B のアドレスには，配列の先頭番地である301番地が割り当てられる。ラベル C は，命令語に変換するアセンブラのラベル表に，304番地とだけ登録される。これを利用すると，ユーザスタック領域を100語分確保するときに，次のようにしてラベルを付けることができる。

```
          DS    100
STACK     DS    0
```

LD命令 LoaD

▶書き方例（R-M形式）

ラベル	命令コード	オペランド
	LD	GR0, ABC
WHO	LD	GR1, BCG, GR2

▶命令の機能

①実効アドレスの記憶内容を，指定したレジスタに設定する。

②LD 命令を実行しても，フラグレジスタ（FR）の内容は変化しない。

▶解説

第1オペランドで，使用するレジスタを指定する。第2オペランド，第3オペランドで，実効アドレスを指定する。第3オペランドがある場合は，第3オペランドで指定した指標レジスタで，アドレス修飾する。

最初のLD 命令を実行すると，ラベル ABC 番地の記憶内容がGR0に設定される。次のLD 命令は，GR2を指標レジスタとしてアドレス修飾する。たとえば，GR2の内容が5とすると，実効アドレスは，

ラベル BCG＋(GR2)＝ラベル BCG＋5＝BCG(5)番地

第2オペランド　第3オペランド

となる。結局，LD 命令によって，配列 BCG(5)番地の記憶内容がGR1に設定される。

LEA 命令では実効アドレスのアドレス値がレジスタに設定されるのに対して，LD 命令では実効アドレスの記憶内容がレジスタに設定される。さらに，LEA 命令ではレジスタに設定した値によりフラグが設定されるのに対して，LD 命令ではフラグは変化しない。

ST命令 STore

▶書き方例（R-M形式）

ラベル	命令コード	オペランド
	ST	GR3, NHK
TBS	ST	1, YTV, 2

▶命令の機能

①指定したレジスタの内容を，実効アドレスが示す記憶場所に格納する。

② ST 命令を実行しても，フラグレジスタ（FR）の内容は変化しない。

▶解説

ST 命令は LD 命令と逆の働きをする。最初の ST 命令では，GR 3 の内容を実効アドレスであるラベル NHK 番地に格納する。

ST 命令を実行しても，フラグレジスタと同様，レジスタの内容も変化しない。

レジスタを指定するには，「GR ＊」のようにフルネームで指定してもよいし，「2」のようにレジスタ番号だけで指定してもよい。

レジスタの記号による指定	汎用レジスタ番号
GR0	0
GR1	1
GR2	2
GR3	3
GR4	4

ラベル TBS 番地の ST 命令を実行すると，GR1の内容が配列 YTV(GR2) 番地に格納される。しかし，GR1とフラグレジスタの内容は変化しない。

LEA命令 Load Effective Address

▶書き方例（R-M形式）

ラベル	命令コード	オペランド	
	LEA	GR0, 0	；(イ)
	LEA	GR1, 1, GR1	；(ロ)
	LEA	GR2, −1, GR2	；(ハ)
	LEA	GR3, ABC	；(ニ)

▶命令の機能

①実効アドレスのアドレス値を，指定したレジスタに設定する。

②命令実行後のレジスタの内容により，フラグレジスタを設定する。

▶解説

行中にセミコロン（；）があると，それ以降，行の終わりまで注釈として扱う（ただし，DC 命令の文字列中の「；」を除く）。なお，第1文字位置に「；」がある場合，または「；」の前に空白しかない場合は，その行全体を注釈として扱う。注釈欄には，処理系で許す任意の文字を書くことができる。

(イ)の LEA 命令の実効アドレスは，0番地であるから，アドレス値0がGR0に設定される。この命令実行後，GR0が0になり，フラグレジスタ FR=「01」となる。

(ロ)の LEA 命令の実効アドレスは，{1＋(GR1)}番地である。そこで，GR1の内容を（＋1）インクリメントしたアドレス値がGR1に設定される。GR0を除くレジスタの内容を（＋1）インクリメントするときに，

LEA　GR＊，1，GR＊　(＊は1〜4)

の命令を使う。

(ハ)の LEA 命令は，(ロ)と逆に(−1)ディクリメントするとき

に使う。GR0は，指標レジスタとして使用できないので，この形式のLEA命令でインクリメント，ディクリメントできない。

(ロ)(ハ)の命令実行後，レジスタの内容によりフラグレジスタが変化する。アドレスが負というのはありえないが，設定されるアドレス値と，その命令実行後のレジスタ，フラグレジスタの関係を表にすると次のようになる。

実効アドレス	レジスタ	フラグ
0	0000000000000000	01
1～32767	0***************	00
32768～65535	1***************	10

(*は0か1)

LD　　GR＊，A　　(＊は0～4)

LEA　　GR＊，0，GR＊　　(＊は1～4)

LEA命令によって，フラグレジスタが変化するのを利用し，上の2命令によって，ラベルA番地に記憶されていた値の正，ゼロ，負の情報をフラグレジスタに設定できる。このときのLEA命令は，NOP (No OPeration: 何もしない)命令だが，フラグレジスタが変化する。このようなNOP命令は，LEA以外の命令でも作れる。

LEA　　GR$，0，GR＊　　($は0～4，＊は1～4)

は，ソースレジスタ(GR＊)の内容をディスティネーションレジスタ(GR$)に転送する，レジスターレジスタ転送命令である。

(ニ)の命令を実行する前，ラベルABC番地が100番地で，その記憶内容が500とすると，LEA命令の実行によって，GR3は100になる。また，LEA命令の代わりに，LD命令を用いると，GR3にはラベルABC番地の記憶内容が設定され，500になる。この2命令の違いをよく理解しておこう。

ADD命令 ADD arithmetic

▶書き方例（R-M形式）

ラベル	命令コード	オペランド	
	ADD	GR0, PC9801	；(イ)
	ADD	GR1, IBM555, GR2	；(ロ)

▶命令の機能

①指定したレジスタの内容と実効アドレスの記憶内容を算術加算し，指定したレジスタに結果を設定する。

②演算結果により，フラグレジスタを設定する。

▶解説

GR0の内容が100で，ラベル PC9801番地の記憶内容が－20のとき，ADD 命令を実行するとGR0の内容は図のように変化する。

フラグレジスタは，演算結果に従って次のように設定される。

(ロ)のADD 命令は，実効アドレスである配列 IBM555 (GR2) の記憶内容を GR1に算術加算する。指標レジスタである GR2の内容を0，1，2……と変化させながら，この ADD 命令を繰り返すことにより，配列データの総和を求めることができる。

SUB命令 SUBtract arithmetic

▶書き方例（R-M形式）

ラベル	命令コード	オペランド
	SUB	GR3, A, GR1

▶命令の機能

①指定したレジスタの内容から実効アドレスの記憶内容を算術減算し，指定したレジスタに結果を設定する。

②演算結果により，フラグレジスタを設定する。

▶解説

GR1の内容が6，GR3の内容が－20，配列要素A(6)の記憶内容が100とすると，GR3の内容－20から実効アドレスであるA(6)番地の記憶内容100を減算し，結果の－120をGR3に設定する。

GR3の内容	1111 1111 1110 1100	－20
－）A(6)番地の記憶内容	0000 0000 0110 0100	100
GR3	1111 1111 1000 1000	－120

一般にコンピュータには減算回路がなく，減数の2の補数を求め，加算回路を使い演算する。

GR3の内容	1111 1111 1110 1100	－20
＋）100の2の補数	1111 1111 1001 1100	－100
GR3	1111 1111 1000 1000	－120

フラグレジスタは，ADD命令と同様，演算結果のレジスタの内容により変化する。

算術演算（ADD，SUB）時に，第0ビットから1がけた上がりするが，オーバフロー（あふれ）ではない。オーバフローとは，レジスタのビット0からのけた上がりを C_0，ビット1からのけた上がりを C_1 とすると，C_0 と C_1 が一致しない状態をいう。

AND命令 AND

▶書き方例（R-M形式）

ラベル	命令コード	オペランド
	AND	GR0, EAD

▶命令の機能

①指定したレジスタの内容と実効アドレスの記憶内容の論理積をとり，指定したレジスタに結果を設定する。

②演算結果により，フラグレジスタを設定する。

▶解説

A	B	A・B
0	0	0
0	1	0
1	0	0
1	1	1

AND（論理積）は図にあるように，両方のビットが「1」のときだけ「1」となり，それ以外は「0」となる。例にあげたAND命令を実行すると次のようになる。

	GR0の内容	1111 0000 1111 0000
AND)	ラベルEAD番地の記憶内容	1111 1111 0000 0000
	GR0	1111 0000 0000 0000

AND命令は，あるビットが「1」か「0」かを調べるときによく使用する。調べたいビットだけを「1」，そのほかのビットを「0」としたデータとANDをとった後，分岐命令で判定する。

フラグレジスタは，ADD命令と同様，演算結果のレジスタの内容により変化する。

論理演算は，符号ビットも含め各ビットごとに処理し，ほかのビットに影響を及ぼさない。

OR命令 OR

▶**書き方例**（R-M形式）

ラベル	命令コード	オペランド
	OR	GR1, CHR, GR3

▶**命令の機能**

①指定したレジスタの内容と実効アドレスの記憶内容の論理和をとり，指定したレジスタに結果を設定する。

②演算結果により，フラグレジスタを設定する。

▶**解説**

A	B	A＋B
0	0	0
0	1	1
1	0	1
1	1	1

OR（論理和）は図にあるように，両方のビットが「0」のときだけ「0」となり，それ以外は「1」となる。

GR3が９のとき，例にあげたOR命令を実行すると，GR1の内容と配列要素CHR(９)の記憶内容の論理和をとり，結果はGR1に設定される。フラグレジスタは，ADD命令と同様，演算結果のレジスタの内容により変化する。

GR1の内容	1111 0000 1111 0000
OR）CHR(GR3)の記憶内容	1111 1111 0000 0000
GR1	1111 1111 1111 0000

論理和（OR）は，算術加算（ADD）と違い，けた上がりはしない。すべての論理演算は各ビットごとに処理し，ほかのビットに影響を及ぼさない。

EOR命令 Exclusive OR

▶書き方例（R-M形式）

ラベル	命令コード	オペランド
	EOR	GR2, TEST
	EOR	GR2, TEST

▶命令の機能

①指定したレジスタの内容と実効アドレスの記憶内容の排他的論理和をとり，指定したレジスタに結果を設定する。

②演算結果により，フラグレジスタを設定する。

▶解説

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

EOR（排他的論理和）は図にあるように，両方のビットパターンが同じときは「0」となり，ビットパターンが異なるときは「1」となる。

GR2の内容	1 0 1 0 1 1 1 1 0 1 0 1 0 0 0 0	
EOR）ラベルTEST番地の記憶内容	1 1 1 1 0 1 0 1 0 0 0 0 1 0 1 0	
GR2	0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0	1回目の結果
EOR）ラベルTEST番地の記憶内容	1 1 1 1 0 1 0 1 0 0 0 0 1 0 1 0	
GR2	1 0 1 0 1 1 1 1 0 1 0 1 0 0 0 0	2回目の結果

レジスタの内容と実効アドレスの記憶内容のビットパターンがまったく同じとき，EOR 命令を実行するとレジスタの内容はオールゼロになる。この性質を利用して，ビットパターンが同じかどうかを調べるときにEOR 命令が使用される。さらに，図にあるように同じEOR 命令を2回実行すると，レジスタの内容がもとに戻る。

CPA命令 ComPare Arithmetic

▶書き方例（R-M形式）

ラベル	命令コード	オペランド
	CPA	GR0, M1

▶命令の機能

指定したレジスタの内容と実効アドレスの記憶内容の算術比較を行い，比較結果によりフラグレジスタに次の値を設定する。

比較結果	フラグレジスタ
(GR＊) ＞ (実効アドレス)	0 0
(GR＊) ＝ (実効アドレス)	0 1
(GR＊) ＜ (実効アドレス)	1 0

(＊は0～4

▶解説

GR0の内容　　　　　　　　　　　　　　ラベルM1番地の記憶内容

0111 1111 1111 1111 ＞ 1111 1111 1111 1111

(32767)　　　　　　　　　　　　　　　(－1)

CPA命令は数値を比較するとき，第0ビットが「1」ならば負数として扱う。図で例のCPA命令を実行すると，FR=「00」になる。

比較命令を実行しても，レジスタの内容および実効アドレスの記憶内容はともに壊れない。比較結果をフラグレジスタに設定する。

```
        CPA     GR0, N50
        JPZ     O50
         :              ;50未満の処理
O50      :
         :              ;50以上の処理
N50     DC      50
```

これで，GR0が50以上か50未満かを区別できる。

CPL命令 ComPare Logical

▶書き方例（R-M形式）

ラベル	命令コード	オペランド
	CPL	GR0, M1

▶命令の機能

指定したレジスタの内容と実効アドレスの記憶内容の論理比較を行い，比較結果によりフラグレジスタに次の値を設定する。

比較結果	フラグレジスタ
(GR*) ＞ (実効アドレス)	0 0
(GR*) ＝ (実効アドレス)	0 1
(GR*) ＜ (実効アドレス)	1 0

(*は0～4)

▶解説

GR0の内容		ラベルM1番地の記憶内容
0111 1111 1111 1111	＜	1111 1111 1111 1111
(32767)		(65535)

CPL命令は数値比較をするとき，単なる2進数として扱うので，オール1の値は－1でなく65535。同じ処理をしても，CPA命令ではFR＝「00」となったが，CPL命令ではFR＝「10」となる。

```
A    LEA   GR0, A
     CPL   GR0, B
      :
B    DC    B
```

CASLでは格納開始アドレスが定義されてないので，A番地の絶対アドレスが32767番地かもしれない。その場合，A番地とB番地のアドレス比較をすると，CPA命令では，B番地のほうが大きいのにB番地が負の値として扱われ，正しい比較ができない。

SLL命令 Shift Left Logical

▶書き方例（R-M形式）

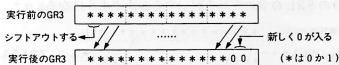
ラベル	命令コード	オペランド
（イ）	SLL	GR3, 2

▶命令の機能

①指定したレジスタの内容を，符号を含め実効アドレスで指定したビット数だけ左にシフトする。シフトの結果，空いたビット位置には0が入る。

②シフトの結果により，フラグレジスタを設定する。

▶解説

書き方例では，GR3の内容を左へ2ビット論理シフトする。

フラグレジスタは，実行後のレジスタの内容により次のように設定される。

算術シフト命令は符号ビット（第0ビット）を除きシフトするので，レジスタの内容が負でも2倍，$\frac{1}{2}$倍の処理がなされる。ところが，論理シフト命令ではそうはいかない。論理シフト命令は，論理演算命令とともにビット処理などに使用される。

SRL命令 Shift Right Logical

▶書き方例（R-M形式）

ラベル	命令コード	オペランド	
	SRL	GR0, 1	；(イ)
	SRL	GR1, 2, GR3	；(ロ)

▶命令の機能

①指定したレジスタの内容を，符号を含め実効アドレスで指定したビット数だけ右にシフトする。シフトの結果，空いたビット位置には0が入る。

②シフトの結果により，フラグレジスタを設定する。

▶解説

(イ)のSRL命令の実行を図示すると次のようになる。

命令実行後のレジスタの内容により，フラグレジスタは変化する。また，16ビットシフトすると，レジスタがオール0になるので，

SRL　GR*，16　(*は0～4)

という命令は，レジスタをオールクリアする場合に使える。

(ロ)の命令を実行する前，GR3の内容が1とすると，インデックス修飾指示があるので，実効アドレスは，

2＋ (GR3)＝2＋1＝3

となる。結局，(ロ)のSRL命令によって，GR1の内容は右へ3ビット論理シフトされる。

SLA命令 Shift Left Arithmetic

▶書き方例（R-M形式）

ラベル	命令コード	オペランド
	SLA	GR0, 1

▶命令の機能

①指定したレジスタの内容を，符号を除き実効アドレスで指定したビット数だけ左にシフトする。シフトの結果，空いたビット位置には0が入る。

②シフトの結果により，フラグレジスタを設定する。

▶解説

例にあげたSLA命令は，GR0の内容を左へ1ビット算術シフトする。これを図示すると次のようになる。

図に示したように，左へ1ビット算術シフトすると，レジスタの内容が正の値でも負の値でも2倍される。左へ2ビット算術シフトすると，4倍，さらに1ビットシフトするごとに，8倍，16倍……となる。このSLA命令は，乗算処理を行うときに用いられる。

フラグレジスタは，SLL命令と同様，実行後のレジスタの内容により変化する。

SRA命令 Shift Right Arithmetic

▶書き方例（R-M形式）

ラベル	命令コード	オペランド
	SRA	GR3, 2

▶命令の機能

①指定したレジスタの内容を，符号を除き実効アドレスで指定したビット数だけ右にシフトする。シフトの結果，空いたビット位置には符号と同じものが入る。

②シフトの結果により，フラグレジスタを設定する。

▶解説

例にあげたSRA命令は，GR3の内容を右へ2ビット算術シフトする。これを図示すると次のようになる。

図に示したように，右へ2ビット算術シフトすると，レジスタの内容が正の値でも負の値でも$\frac{1}{4}$倍される。もちろん右へ1ビット算術シフトすると，$\frac{1}{2}$倍したことになる。

レジスタの内容が負の場合，右へ16ビット算術シフトしてもゼロにならず，−1になることに注意。

フラグレジスタは，SLL命令と同様，実行後のレジスタの内容により変化する。

JPZ命令 Jump on Plus or Zero

▶書き方例（M形式）

ラベル	命令コード	オペランド
	JPZ	PLUSZ

▶命令の機能

フラグレジスタの値がFR＝「00」あるいはFR＝「01」のとき，実効アドレスに分岐する。それ以外のとき(FR＝「10」)は，次の命令に進む。

▶解説

JMI命令と逆の処理を行う。例にあげたJPZ命令の流れを図示すると次のようになる。

```
      JPZ   PLUSZ
┌────────────────────────┐
│        ⋮        ；＜0  │ 負の処理ブロック
└────────────────────────┘
┌────────────────────────┐
│PLUSZ   ⋮        ；≧0  │ 正，ゼロの処理ブロック
└────────────────────────┘
```

ゼロのときに負と同じ処理を行うためには，次のようにする。

```
      JZE   MZ
      JPZ   P
┌────────────────────────┐
│MZ      ⋮        ；≦0  │ 負，ゼロの処理ブロック
└────────────────────────┘
┌────────────────────────┐
│P       ⋮        ；＞0  │ 正の処理ブロック
└────────────────────────┘
```

さらに，正，ゼロ，負ごとに3つの処理に分けるには，次のようにする。

```
      JZE   Z
      JPZ   P
┌────────────────────────┐
│        ⋮        ；＜0  │ 負の処理ブロック
└────────────────────────┘
┌────────────────────────┐
│Z       ⋮        ；＝0  │ ゼロの処理ブロック
└────────────────────────┘
┌────────────────────────┐
│P       ⋮        ；＞0  │ 正の処理ブロック
└────────────────────────┘
```

JMI命令 Jump on MInus

▶書き方例（M形式）

ラベル	命令コード	オペランド
	JMI	MINUS

▶命令の機能

フラグレジスタの値がFR＝「10」のとき，実効アドレスに分岐する。それ以外のとき（FR＝「00」あるいはFR＝「01」）は，次の命令に進む。

▶解説

JPZ命令と逆の処理を行う。例にあげたJMI命令の流れを図示すると次のようになる。

```
        JMI     MINUS
┌──────────────────────────┐
│       ⋮          ; ≧0   │ 正，ゼロの処理ブロック
└──────────────────────────┘
┌──────────────────────────┐
│ MINUS ⋮          ; <0   │ 負の処理ブロック
└──────────────────────────┘
```

ゼロのとき負と同じ処理を行うためには，次のようにする。

```
        JMI     MZ
        JZE     MZ
┌──────────────────────────┐
│       ⋮          ; >0   │ 正の処理ブロック
└──────────────────────────┘
┌──────────────────────────┐
│ MZ    ⋮          ; ≦0   │ 負，ゼロの処理ブロック
└──────────────────────────┘
```

さらに，正，ゼロ，負ごとに3つの処理に分けるには，次のようにする。

```
        JMI     M
        JZE     Z
┌──────────────────────────┐
│       ⋮          ; >0   │ 正の処理ブロック
└──────────────────────────┘
┌──────────────────────────┐
│ Z     ⋮          ; =0   │ ゼロの処理ブロック
└──────────────────────────┘
┌──────────────────────────┐
│ M     ⋮          ; <0   │ 負の処理ブロック
└──────────────────────────┘
```

JNZ命令 Jump on Non Zero

▶書き方例（M形式）

ラベル	命令コード	オペランド
	JNZ	NZERO

▶命令の機能

フラグレジスタの値がFR＝「00」あるいはFR＝「10」のとき，実効アドレスに分岐する。それ以外のとき（FR＝「01」）は，次の命令に進む。

▶解説

JZE命令と逆の処理を行う。例にあげたJNZ命令の流れを図示すると次のようになる。

```
        JNZ    NZERO
+--------------------------------------+
|             ⋮              ; = 0     |   ゼロの処理ブロック
+--------------------------------------+
| NZERO       ⋮              ; ≠ 0     |   非ゼロの処理ブロック
+--------------------------------------+
```

JZE命令 Jump on ZEro

▶書き方例（M形式）

ラベル	命令コード	オペランド
	JZE	ZERO

▶命令の機能

フラグレジスタの値がFR=「01」のとき，実効アドレスに分岐する。それ以外のとき（FR=「00」あるいはFR=「10」）は，次の命令に進む。

▶解説

JNZ命令と逆の処理を行う。例にあげたJZE命令の流れを図示すると次のようになる。

```
        JZE     ZERO
┌─────────────────────┬──────┐ ┌─────────────────┐
│           ⋮         │ ；≠0 │ │非ゼロの処理ブロック│
└─────────────────────┴──────┘ └─────────────────┘
┌─────────────────────┬──────┐ ┌─────────────────┐
│ZERO       ⋮         │ ；=0 │ │ゼロの処理ブロック  │
└─────────────────────┴──────┘ └─────────────────┘
```

JMP命令 unconditional JuMP

▶書き方例（M形式）

ラベル	命令コード	オペランド	
	JMP	ALL	;（イ）
	JMP	INDEX, GR1	;（ロ）

▶命令の機能

無条件に実効アドレスに分岐する。

▶解説

例の(イ)にあげたJMP命令を実行すると，実効アドレスであるラベルALL番地に無条件にジャンプする。ハードウェア的に命令を説明すると，実効アドレスをプログラムカウンタ（Program Counter：PC）に設定する。

例の(ロ)にあげたインデックス修飾のあるJMP命令では，指標レジスタの内容によって分岐先が変化する。

ブロック処理が終了した後，無条件ジャンプを最後に入れる必要がある。

	JPZ PLUSZ	
⋮	; < 0	負の処理ブロック
	JMP NEXT	次の処理ブロックへ分岐
PLUSZ ⋮	; ≧0	正，ゼロの処理ブロック
NEXT ⋮		次の処理ブロック

PUSH命令 PUSH effective address

▶書き方例（M形式）

ラベル	命令コード	オペランド	
	PUSH	12345	;（イ）
	PUSH	0, GR1	;（ロ）

▶命令の機能

①スタックポインタから1をアドレス減算する。

②実効アドレスをスタックポインタの内容の示す記憶場所に格納する。

▶解説

GR4がスタックポインタ（stack pointer）として用いられる。スタックポインタは，スタックの最上段のアドレスを保持しているレジスタである。いま，GR4の内容が800のとき，例にあげた（イ）のPUSH 命令を実行すると，799番地に12345が格納される。

つづいて，例にあげた（ロ）のPUSH 命令を実行すると，実効アドレスである{0＋(GR1)}番地が798番地に格納される。この（ロ）の命令は，GR1～GR4の内容をスタック領域に退避するときに使う命令である。逆にスタック領域から値を取り出すには，POP 命令を使う。

普通の場合，命令によって主記憶装置にデータを記憶させるには，何番地に記憶せよというように，常に実効アドレスを示さなければならない。ところが，スタックに記憶させるには，番地を指定しなくてもよい。ここが大きな違いである。

POP命令 POP up

▶書き方例（その他の形式）

ラベル	命令コード	オペランド
	POP	GR1

▶命令の機能

①スタックポインタの内容が示す記憶場所から記憶内容を取り出し，指定したレジスタに設定する。

②スタックポインタに1をアドレス加算する。

▶解説

PUSH 命令の逆の処理を行う。もちろん，GR4がスタックポインタで，スタックの最上段のアドレスを保持している。いま，GR4の内容が798のとき，例にあげたPOP 命令を実行すると，798番地の記憶内容がGR1に設定される。主プログラムから渡されたレジスタの内容を保持するために，副プログラムでは次のようにする。

```
SUBPRO   START
         PUSH    0，GR1
         PUSH    0，GR2
         PUSH    0，GR3
           ⋮
         POP     GR3
         POP     GR2
         POP     GR1
         RET
         END
```

このとき，PUSH するレジスタと POP するレジスタの順が逆であることに注意しよう。

CALL命令 CALL subroutine

▶書き方例（M形式）

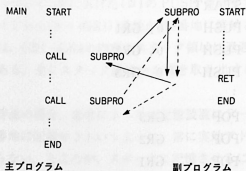
ラベル	命令コード	オペランド
	CALL	SUBPRO

▶命令の機能

①プログラムカウンタの現在値に2をアドレス加算した値をスタックにPUSHする。

②実効アドレスをプログラムカウンタに設定する。

▶解説

例にあげたCALL命令を簡単に説明すれば，副プログラムSUBPROに分岐することである。ただ単に分岐するだけならば，無条件ジャンプ命令と同じになる。JMP命令と異なる点は，戻り番地（CALL命令の次の命令が記憶されている番地）をスタックにPUSHしてから分岐することである。

図に示したように，CALL命令は，RET命令とペアで使い，副プログラムに分岐し，主プログラムの呼ばれた次の命令に戻る。

RET命令 RETurn from subroutine

▶書き方例（その他の形式）

ラベル	命令コード	オペランド
	RET	

▶命令の機能

スタックポインタからPOPした値をプログラムカウンタに設定する。

▶解説

RET 命令は，CALL 命令で呼ばれた戻り番地(CALL 命令の次の命令を記憶している番地）に戻る命令。なお，戻り番地はスタックに格納されている。

プログラムカウンタ（PC）は，実行中の命令語の先頭アドレスを保持し，命令の実行が終わると，次に実行する命令語の先頭アドレスが設定される。一般に，命令の実行が終わるとPCに2がアドレス加算され，分岐，コール，リターン命令の場合は，新たに分岐先のアドレスが設定される。

PCを汎用レジスタのように書くことができれば，

```
CALL    SUBPRO
```

という命令は，次の2命令と同じ意味になる。

```
PUSH    2, PC        ⎫
LEA     PC, SUBPRO   ⎭……＊＊
```

(LEA 命令は「JMP SUBPRO」でもよい)

また，次の2命令は同じ意味である。

```
RET
POP     PC    ……＊＊
```

＊＊のような命令はないが，CALL，RET 命令を理解しやすい。

IN命令 INput

▶書き方例（その他の形式）

ラベル	命令コード	オペランド
	IN	INBUF, NCHR

▶命令の機能

①入力領域に1レコードの文字データを入力する。

②入力領域は，第1オペランドにラベル名で指定し，入力レコードの文字数を第2オペランドにラベル名で指定する。

③入力領域は，80語長（80文字分）の作業域のラベル名とし，この領域に先頭番地から1文字を1語に対応させて，順次入力する。各語の第0～7ビットには，0を格納する。

④入力レコードの文字数を，第2オペランドで指定した領域（1語）に2進データの形で格納する。レコードの区切符号（鍵盤入力のときの復帰符号など）は，格納しない。

⑤入力データが80語に満たない場合，入力領域の残りの部分は実行前のデータを保持する。入力データが80文字を超える場合，それ以降の文字は無視する。

⑥次の場合，第2オペランドで指定した領域（入力文字長）に，0または－1を格納する。

0――空のレコードの入力（タイプライタで復帰符号だけが入力されたときなど）

－1――EOF（End Of File）の検出（カード読取り装置など）

⑦GRの内容は保存されるが，FRの内容は不定となる。

▶解説

図に示すように，3枚のカードが例にあげたIN命令によって入力された場合を考える。1枚目のカードが入力された後，6文字入

力されたので，ラベルNCHR番地には2進数の6が格納される。配列IBUF番地には，入力された文字の文字コードが1文字ごとに格納される。

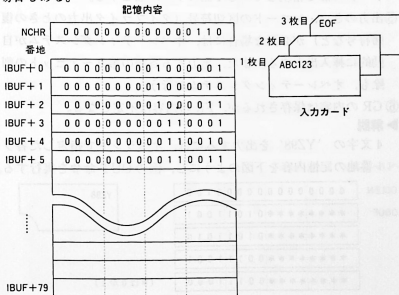
2枚目のカードが例にあげたIN命令によって入力されると，カードは空白行なのでラベルNCHR番地が0になる。

ラベルNCHR番地　0000 0000 0000 0000

配列IBUFの内容は，実行前のデータが保持されるので変わらない。

3枚目のカードについてIN命令を実行すると，EOFなのでラベルNCHR番地が－1になる。

ラベルNCHR番地　1111 1111 1111 1111

もちろん，配列IBUFの内容は変わらない。なお，EOFは，処理系によって，「//EOD.」とか「//EOF.」などの制御カードを入れる場合もある。

OUT命令 OUTput

▶書き方例（その他の形式）

ラベル	命令コード	オペランド
	OUT	OBUF, OCLEN

▶命令の機能

①第1オペランドのラベル名で指定した出力領域に格納されている文字データを，1レコードとして出力する。

②第2オペランドのラベル名で出力文字長を指定する。

③出力領域は，出力しようとするデータが1文字1語で格納されている領域のラベル名とする（DC命令の文字定数に同じ。ただし，第0～7ビットの値は0でなくてもよい）。

④出力文字長は，1レコードとして出力しようとする文字数を2進データの形で格納している1語のラベル名とする。

⑤出力のさい，レコードの区切符号（タイプライタ出力のときの復帰符号など）が必要な場合には，オペレーティングシステムが自動的に挿入出力する。また，出力する各語の第0～7ビットの削除も，オペレーティングシステムが行う。

⑥GRの内容は保存されるが，FRの内容は不定となる。

▶解説

4文字の‘YZ98’を出力するには，オペランドで指定した各ラベル番地の記憶内容を下図のようにし，例のOUT命令を実行する。

（＊は0か1）

EXIT命令 EXIT

▶ **書き方例**（その他の形式）

ラベル	命令コード	オペランド
	EXIT	

▶ **命令の機能**

①プログラムの実行を終了する（制御をオペレーティングシステムに戻す）。

② GR の内容は保存されるが，FR の内容は不定となる。

▶ **解説**

オペランド欄は空白にする。

EXIT 命令は命令語に変換される。FORTRAN 言語の STOP 命令と同じだが，COMET が停止するのではなく，オペレーティングシステムの先頭に戻る（ジャンプする）。

■チェック問題1

CASL 言語で書かれた次のプログラムについて，各設問に答えよ。

行番号	ラベル	命令コード	オペランド
10	EX1	START	
20		LD	GR1, A
30		ADD	GR1, B
40		SUB	GR1, C
50		JPZ	L1
60		EOR	GR1, M1
70		LEA	GR1, 1, GR1
80	L1	ST	GR1, ANS
90		EXIT	
100	A	DC	100
110	B	DC	200
120	C	DC	500
130	M1	DC	−1
140	ANS	DS	1
150		END	

設問1　プログラム EX1 の実行が終了すると，ラベル ANS 番地の記憶内容はいくらになるか。

設問2　行番号60，70の2命令によって，どのような処理がされるか。

設問3　ラベルA. B. C番地の記憶内容を−100，−200，−500と変更してからプログラム EX1 を実行すると，実行終了後ラベル ANS 番地の記憶内容はいくらになるか。4けた16進数で答えよ。

■ チェック問題1解答

設問1　200

設問2　2の補数を求める処理

設問3　X "00C8 "（ただし，X "　"は16進数を示す）

【重要ポイント】　2の補数を求めるには，NOT して(＋1)する。NOT (論理否定) は，オール1のデータと EOR する。

■チェック問題❷

CASL 言語で書かれた次のプログラムについて，各設問に答えよ。

行番号	ラベル	命令コード	オペランド
10	EX2	START	PROG
20	CIN	DS	80
30	NIN	DS	1
40	C0	DC	'0'
50	OUTC	DC	'LOOP␣'
60	OUTL	DS	1
70	OUTN	DC	6
80	N1	DC	1
90	PROG	IN	CIN, NIN
100		OUT	CIN, NIN
110		LD	GR0, CIN
120	AGN	EOR	GR0, C0
130		JZE	STOP
140		EOR	GR0, C0
150		ST	GR0, OUTL
160		OUT	OUTC, OUTN
170		SUB	GR0, N1
180		JMP	AGN
190	STOP	EXIT	
200		END	

設問 1　データとして 3 を入力してこのプログラム EX2 を実行すると，どのような出力が得られるか。

設問 2　行番号120，140の EOR 命令の役割を簡単に説明せよ。

■ チェック問題❷解答

設問 1

```
3
LOOP␣3
LOOP␣2
LOOP␣1
```

設問 2

行番号120　文字コードを数値に変換する
行番号140　数値を文字コードに変換する

【重要ポイント】 IN，OUT 命令によって入出力されるデータは，文字コードである。文字コードを数値に変換する方法をマスターしよう。

■チェック問題3

CASL 言語で書かれた次のプログラムは，GR1（上位16ビット）と GR2（下位16ビット）を連続した32ビットのレジスタと考えて，1ビット左（SLDA）あるいは右（SRDA）に算術シフトする副プログラムである。空欄に適切な命令を入れよ。

行番号	ラベル	命令コード	オペランド
10	SLDA	START	
20		SLA	GR1, 1
30		SLA	GR2, 0
40		JPZ	SLDA1
50		[ ]	
60	SLDA1	[ ]	
70		RET	
80		END	
90			
100	SRDA	START	
110		PUSH	0, GR1
120		SRL	GR2, 1
130		AND	GR1, X0001
140		JZE	SRDA1
150		[ ]	
160	SRDA1	[ ]	
170		SRA	GR1, 1
180		RET	
190	X0001	DC	#0001
200		END	

■ チェック問題3解答

行番号 50　LEA　GR1, 1, GR1
行番号 60　SLL　GR2, 1
行番号150　LEA　GR2, 32768, GR2
行番号160　POP　GR1

【重要ポイント】あるビットを立てるのに，LEA 命令を利用する。
AND すればレジスタの前の内容が消えてしまう。

■チェック問題4

CASL言語で書かれた次のプログラムは，*N*個のデータ（*DATA*(0)～*DATA*(*N*−1)）の総和，最大値，最小値を GR0,GR2,GR3 にそれぞれ求めるプログラムである。空欄に適切な命令を入れよ。

行番号	ラベル	命令コード	オペランド
10	EX4	START	
20		[　　　　]	
30		LEA	GR1, 0
40		LD	GR2, DATA
50		LD	GR3, DATA
60	L1	CPA	GR1, N
70		[　　　　]	
80		EXIT	
90	L2	ADD	GR0, DATA, GR1
100		CPA	GR2, DATA, GR1
110		JPZ	L3
120		[　　　　]	
130		[　　　　]	
140	L3	CPA	GR3, DATA, GR1
150		JMI	L4
160		LD	GR3, DATA, GR1
170	L4	[　　　　]	
180		JMP	L1
190	N	DS	1
200	DATA	DS	100
210		END	

■ チェック問題4解答

行番号　20　LEA　GR0, 0
行番号　70　JMI　L2　（JNZ　L2でも可）
行番号 120　LD　GR2, DATA, GR1
行番号 130　JMP　L4
行番号 170　LEA　GR1, 1, GR1

【重要ポイント】 大小比較を CPA，CPL 命令で行うと，レジスタの内容も記憶装置の記憶内容も壊れない。

■チェック問題5

CASL 言語で書かれた次のプログラムは，主プログラムの CALL 命令の次の 2 語に記憶されている値を加算し，加算結果を GR0 に求めるプログラムである。各設問に答えよ。

行番号	ラベル	命令コード	オペランド
10	EX5	START	
20		CALL	WA
30		DC	1234
40		DC	2345
50		ST	GR0, ANS1
60		CALL	WA
70		DC	−3456
80		DC	−4567
90		ST	GR0, ANS2
100		EXIT	
110	ANS1	DS	1
120	ANS2	DS	1
130		END	
140			
150	WA	START	
160		POP	GR1
170			
180		ADD	GR0, 1, GR1
190			
200		RET	
210		END	

設問 1　空欄に適切な命令を入れよ。

設問 2　プログラム EX5 の実行が終了したとき，ANS1，ANS2 番地の記憶内容はいくらか。

■チェック問題5解答

設問 1　行番号170　LD　GR0，0，GR1
　　　　行番号190　PUSH　2，GR1

設問 2　ANS 1 番地　3579　ANS 2 番地　−8023

【重要ポイント】 副プログラムにきたとき，スタックの頭に戻り番地がある。これを取り出して引数の処理ができる。RET する前に戻り番地を整える。

和文索引 (50音順)

あ―お

か―こ

さ―そ

た―と

な―の

は―ほ

ま―も

ら―ろ

欧文索引

FORTRAN

基本文法ハンドブック

情報処理技術インストラクタ
滝沢英子

▶このFORTRAN基本文法ハンドブックは，FORTRANの初心者から中級者までを対象に，プログラミング上の規則や基本コマンドの使い方などを解説したものです。

▶文法事項は，日本工業規格JIS FORTRAN C6201の上位水準に準拠しています。ただし，第2種情報処理技術者試験の範囲から除かれた次の仕様についての説明は省いてあります。

倍精度実数型，複素数型データ
内部ファイル，INQUIRE文

▶本文中で示したFORTRANの文などの一般形式に含まれる記号類は，次の意味を表します。

①ブラケット［　］のなかは指定しなくてもよい

②リーダー…はその前の指定項目を繰り返し指定してもよい

FORTRAN
基本文法ハンドブック—目次

FORTRANプログラムの構成

〔プログラム例〕

50人の学生について，3科目のテストの平均点を求める。

```
*      SAMPLE PROGRAM                               注釈行
       CHARACTER*20 NAME                            型宣言文
       WRITE(6,1)                                   WRITE文
     1 FORMAT('1',10X,'** LIST **'/               ┐
      * '0',4X,'NAMAE',17X,'TEST',                ├FORMAT文
      *     7X,'HEIKIN')                          ┘
       DO 10 I=1,50                                 DO文
         READ(5,2) NAME,K1,K2,K3                    READ文
     2   FORMAT(A20,3I3)                            FORMAT文
         H=(K1+K2+K3)/3.0                           算術代入文
         WRITE(6,3) NAME,K1,K2,K3,H                 WRITE文
     3   FORMAT(' ',A20,3I5,F8.1)                   FORMAT文
    10   CONTINUE                                   CONTINUE文
         STOP                                       STOP文
         END                                        END文
```

①注釈行と継続行

第１けたがＣまたは＊の行は注釈行(プログラムの説明)となる。

第６けたに０でない文字のある行は継続行といい，前の行からつながっているとみなされる。

②文と文番号

第７けたから72けたの間に文を書く。文の識別のために第１けたから５けたの間に番号を付けることができる。これを文番号という。

③実行文と非実行文

実行文は動作を指定するための文であり，基本的には先頭から順に実行される。プログラム中のWRITE文，DO文，READ文，算術代入文，CONTINUE文，STOP文，END文は実行文である。

宣言のための文や入出力の編集情報のための文は非実行文である。型宣言文やFORMAT文がこれに当たる。

定数と変数

▶定数

整数型	1, 5, −10
実数型	0.5, −10.3, 1.5E6, −12E−3
文字型	'ABC', 'I DON''T KNOW'
論理型	.TRUE., .FALSE.

プログラム中に直接，指定するものを**定数**という。数値データは，小数点の付かないものを整数型，小数点の付いたものを実数型の定数として扱う。実数型の定数は指数部付きのものも許され，Eの後ろに指定する。1.5E6は1.5×10^{6}，−12E−3は-12×10^{-3}。

文字データは前後にアポストロフィを付けて定数とする。文字データ中にアポストロフィが含まれる場合，アポストロフィ1つに対しアポストロフィを2つ並べて指定する。'I DON''T KNOW'は，I DON'T KNOWという文字データを表す。論理型の定数には，真を表す.TRUE.と偽を表す.FALSE.の2とおりしかない。

▶変数

変数名は6文字以内の英数字。先頭の文字は英字。

整数型　先頭がI, J, K, L, M, Nの変数

実数型　先頭がI, J, K, L, M, N以外の変数

このほかに型の宣言を行えば，文字型，論理型を含めて自由に型を指定できる。

定数に対し，プログラム作成者が名前を付けて扱うのが**変数**である。変数の値は実行時にいろいろ変化する。定数と同様，型があり，その記憶する数の表現形式により，変数名で型の宣言をする必要がある。ただし，整数型と実数型については，上記のとおり変数名の先頭文字で区別することができる。これを**暗黙の型宣言**という。

配　列

▶配列の宣言

配列を使用するときは，DIMENSION 文または型宣言文に配列宣言子を指定する（仮配列の宣言は p. 42参照）。

配列宣言子　***a*** (***d*** [, ***d***] …)
a は配列名，*d* は寸法宣言子。*d* の個数を配列の次元数という。

寸法宣言子　[d_1 :] d_2
d_1には寸法の下限，d_2には寸法の上限を指定する。d_1が1のときには d_2だけでよい。d_2-d_1+1を寸法の大きさという。 d_1，d_2を寸法の限界式といい，整定数式（定数と定数名だけを含む結果が整数型の式）が指定できる。

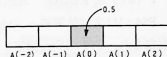
▶配列要素の参照

配列要素名　***a*** (***s*** [, ***s***] …)
a は配列名，(*s* [, *s*] …) は添字。*s* は添字式で整数式（結果が整数型の算術式）を指定する。

〔例1〕 1次元の配列

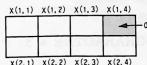
```
DIMENSION  A (-2:2)
    ⋮
A (0)=0.5
```

0.5

A(-2)	A(-1)	A(0)	A(1)	A(2)
		0.5		

〔例2〕 2次元の配列

```
DIMENSION  X (2, 4)
INTEGER  X
X (1, 4)=0
```

X(1,1)	X(1,2)	X(1,3)	X(1,4)
X(2,1)	X(2,2)	X(2,3)	X(2,4)

0 → X(1,4)

型宣言文で配列の宣言も行えるので，〔例2〕の宣言は「INTEGER　X(2, 4)」のようにまとめられる。

部分列

▶文字部分列

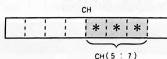
v ([e_1]：[e_2])
　または，
a (*s* [, *s*] …) ([e_1]：[e_2])

v は文字型の変数名，*a* (*s* [, *s*] …) は文字型の配列要素名とする。

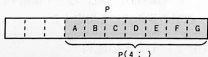
e_1，e_2は文字位置式といい，整数式が指定できる。e_1が1のときにはe_1を省略できる。また，e_2が *v* または *a* (*s* [, *s*] …) の長さと等しいときはe_2を省略できる。

文字型の変数や配列要素の一部分を表すときに，文字部分列を使用する。

〔例1〕

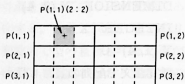
```
CHARACTER * 8  CH
CH (5: 7) ='* * *'
```


〔例2〕

```
CHARACTER * 10  P
P (4: ) ='ABCD'//'EFG'
```


P(4：)は文字位置式 e_2 の部分が省略されているので，Pの長さと仮定され，P(4：10)と指定したのと同じ意味になる。// は文字列の連結の演算子。

〔例3〕

```
CHARACTER * 3  P (3, 2)
        ⋮
P (1, 1) (2:2) ='+'
```

式

▶式の種類

算術式　算術演算を表すのに用いる。結果として数値を得る。

文字式　文字列を表すのに用いる。文字型の結果を得る。

関係式　2つの算術式の値または2つの文字式の値を比較するのに用いる。算術式の値と文字式の値を比較してはならない。真または偽の値をもつ論理型の結果を得る。

論理式　論理演算を表すのに用いる。真または偽の値をもつ論理型の結果を得る。

各式を構成するための演算子として，**算術演算子**，**文字演算子**，**関係演算子**，**論理演算子**の4種類がある。

▶演算子の種類と働き

種　類	演算子	意　味	用　法	解　釈
算術演算子	**	べき乗	x_1 ** x_2	x_1をx_2乗する
	/	除算	x_1/x_2	x_1をx_2で割る
	*	乗算	x_1 * x_2	x_1とx_2を掛ける
	−	減算または符号逆転	x_1 − x_2	x_1からx_2を引く
			− x_1	x_1の符号を逆転する
	+	加算または同値	x_1 + x_2	x_1とx_2を加える
			+ x_1	x_1と同じ
文字演算子	//	連結	x_1//x_2	x_1とx_2を連結する
関係演算子	.LT.	小さい（<）	x_1.LT.x_2	$x_1 < x_2$のとき真 それ以外偽
	.LE.	小さいか等しい(≦)	x_1.LE.x_2	$x_1 \leqq x_2$　〃
	.EQ.	等しい（＝）	x_1.EQ.x_2	$x_1 = x_2$　〃
	.NE.	等しくない（≠）	x_1.NE.x_2	$x_1 \neq x_2$　〃
	.GT.	大きい（>）	x_1.GT.x_2	$x_1 > x_2$　〃
	.GE.	大きいか等しい(≧)	x_1.GE.x_2	$x_1 \geqq x_2$　〃

種　類	演算子	意　味	用　法	解　釈
論理演算子	.NOT.	否定	.NOT. x_1	(注)
	.AND.	論理積	x_1.AND.x_2	
	.OR.	論理和	x_1.OR.x_2	
	.EQV.	論理等価	x_1.EQV.x_2	
	.NEQV.	論理非等価	x_1.NEQV.x_2	

（注）　論理演算子の働きは次のとおり。

x_1	.NOT.x_1
真	偽
偽	真

x_1	x_2	x_1.AND.x_2	x_1.OR.x_2	x_1.EQV.x_2	x_1.NEQV.x_2
真	真	真	真	真	偽
真	偽	偽	真	偽	真
偽	真	偽	真	偽	真
偽	偽	偽	偽	真	偽

演算子には次のような演算の優先順位がある。これを変更したければ括弧(　)を用いる。

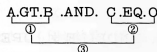
優先順位	種　類	演 算 子
1	算術演算子	** *, / +, −
2	文字演算子	//
3	関係演算子	.LT., .LE., .EQ., .NE., .GT., .GE.

優先順位	種　類	演 算 子
4	論理演算子	.NOT. .AND. .OR. .EQV.,.NEQV.

同じ優先順位の枠内では，上から下へ高位→低位となる。

〔例〕演算順序を①②…で示す。

A+B*C　　(A+B)*C

A.GT.B .AND. C.EQ.O　　A//B.EQ.C

算術式に整数型と実数型が混在している場合，整数型のデータが実数型にそろえられてから演算される。結果は実数型で求められる。整数型どうしの演算では結果も整数型になる（特に除算の場合注意）。

〔例〕 次のような宣言があった場合で考える。

REAL　　　A, X（変数 A と X は実数型として宣言）

INTEGER　M,N（変数 M と N は整数型として宣言）

A+X*N　　N の値が実数化され，①の演算が行われる。

A+M/2　　①の除算は整数型どうしなので，結果が整数型で求められる（切捨て）。M が 1 の場合は1/2の値は 0 なので，この算術式の値は A の値となる。

文の種類

文には実行文と非実行文がある。以下に文の種類をあげる。

▶実行文

①算術代入文，論理代入文，文番号代入文（ASSIGN 文），文字代入文

②単純 GO TO 文，計算形 GO TO 文，割当て形 GO TO 文

③算術 IF 文，論理 IF 文

④ブロック IF 文，ELSE 文，END IF 文，ELSE IF 文

⑤ CONTINUE 文

⑥ STOP 文，PAUSE 文

⑦ DO 文

⑧ READ 文，WRITE 文，PRINT 文

⑨ REWIND 文，BACKSPACE 文，ENDFILE 文，OPEN 文，CLOSE 文

⑩ CALL 文，RETURN 文

⑪ END 文

▶非実行文

① PROGRAM 文，FUNCTION 文，SUBROUTINE 文，ENTRY 文，BLOCK DATA 文

② DIMENSION 文，COMMON 文，EQUIVALENCE 文，IMPLICIT 文，PARAMETER 文，EXTERNAL 文，INTRINSIC 文

③ INTEGER 文，REAL 文，LOGICAL 文，CHARACTER 文

④ DATA 文

⑤ FORMAT 文

⑥文関数定義文

宣言文

▶ EQUIVALENCE 文

EQUIVALENCE (*nlist*)　[, (*nlist*)] …

nlist は変数名，配列要素名，文字部分列の並び。各並びは少なくとも 2 つの名前を含むこと。また，*nlist* の並びのなかにある添字式や文字位置式は整定数式でなければならない。

機能►►1 つの *nlist* 中に現れる名前が，すべて同じ記憶場所にあるものと指示するための文。

A(1)　A(2)　A(3)
X

① DIMENSION　A (3),　B (3, 2)
　EQUIVALENCE　(A (3), X)

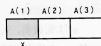
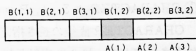
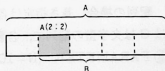
A(3)と X が同じ記憶場所になる。A(3)=0 とする代わりに X=0 としても同じ結果が得られる。

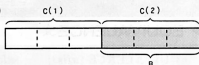
EQUIVALENCE　(A, X)

A(1)　A(2)　A(3)
X

配列名を指定した場合，その配列の先頭の要素を指定したことと同じ。

EQUIVALENCE (A, B (1, 2))

B(1,1)　B(2,1)　B(3,1)　B(1,2)　B(2,2)　B(3,2)
A(1)　A(2)　A(3)

2 次元の配列では，要素は列ごとに並んでいると考える。

② CHARACTER　A ∗ 5, B ∗ 3, C (2)∗ 3
　EQUIVALENCE　(A (2 : 2), B)

文字型の場合は文字型のものとしか記憶場所を共有できない。

EQUIVALENCE (B, C (2))

▶型宣言文

typ v [, *v*] …
typ は INTEGER, REAL, LOGICAL のいずれか。*v* は変数名，配列名，配列宣言子，定数名，関数名，または仮手続き名とする。

機能►►文字型以外のデータの型を宣言する。*typ* が INTEGER だと整数型，REAL は実数型，LOGICAL は論理型となる。

CHARACTER [* *len* [,]] *nam* [, *nam*] …
nam は次のいずれか。 *v* [* *len*]　　　*a* [(*d*)] [* *len*] *v* は変数名，定数名，関数名または仮手続き名とする。*a* は配列名，*a* (*d*) は配列宣言子。*len* は長さ（文字の個数）で，これを長さ指定という。省略すると 1 となる。

機能►►データが文字型であることを宣言する。

① CHARACTER＊3　A, B＊5

CHARACTER 直後の長さ指定は，変数名や配列宣言子などの後に長さ指定がない場合に適用される。①では A は 3，B は 5 の長さの文字型として宣言される。

② CHARACTER　C (2)＊3

配列の場合，長さ指定は各配列要素に対してのものになる。②では C は文字型の配列で，各要素 C(1)，C(2)は長さ 3 となる。

③ PARAMETER　(N=10)

CHARACTER　X＊(N+1)

長さ指定には整定数式が指定できる。この場合，式は括弧でくくること。③のXは長さ11文字となる。

④CHARACTER *(*)　CH

PARAMETER　(CH='ABC')

文字型の定数名の場合，長さ指定に（*）が使用できる。この場合，PARAMETER文で設定した文字定数の長さになる。④ではCHは長さ3の文字型の定数。このほか，文字型の仮引数の宣言にも(*)の長さ指定が使用できる。この場合，対応する実引数の長さと合わせられる。

▶IMPLICIT文

IMPLICIT ***typ*** (*a* [, *a*] …) [, ***typ*** (*a* [, *a*] …)] …
typ はINTEGER，REAL，LOGICAL，CHARACTER[* *len*]のいずれか。*a* は1つの英字またはアルファベット順での英字の範囲。

機能▶▶指定された英字で始まる名前の変数，配列，定数名，外部関数および文関数の型を宣言する。

データの型の決定方法には，型宣言文，IMPLICIT文，暗黙の型宣言があるが，優先順位は次のとおり。

(i)型宣言文

(ii)IMPLICIT文

(iii)暗黙の型宣言

①IMPLICIT　REAL (L), INTEGER (X-Z)

英字の範囲で宣言する場合，英字を－でつなぐ。①ではLで始まる変数や配列などはすべて実数型，X，Y，Zで始まるものは整数型になる。

②IMPLICIT　CHARACTER * 10　(F)

INTEGER　FREQ

Fで始まる名前は10文字の文字型であると宣言されているが，型

宣言文でFREQだけは整数型として宣言し直している。

▶ PARAMETER文

PARAMETER (*p*=*e* [, *p*=*e*] …)
*p*を定数名という。名称の付け方の規則は変数名と同じ。*e*には定数を指定する。

機能►►定数に対し名前を付ける。

① PARAMETER (N=5)
　DIMENSION X(−N : N)
　このDIMENSION文はDIMENSION X(−5 : 5)と同じ。

② CHARACTER * 4 P
　PARAMETER (P= * * * *)

初期値の設定

▶ DATA文

DATA *nlist*/*clist*/ [[,] *nlist*/*clist*/] …
nlist は変数名，配列名，配列要素名，部分列名またはDO形並びの並び。*clist* は「*c*」と「*r* ＊ *c*」の形の並び。*c* は定数または定数名。*r* はゼロでない符号なし整定数，またはこのような定数の定数名。

▶ DO形並び

(*dlist*, $i=m_1$, m_2 [, m_3])
dlist は配列要素名またはDO形並び。*i* は整数型の変数名でDO形制御変数という。m_1，m_2，m_3は整定数式。

機能►►変数や配列などに初期値を設定する。ただし，「仮引数」，「無名共通ブロック中の要素」においては初期値設定を行えない。

① DATA　A, B/1.0, 2.5/

nlist 中の項目は，*clist* 中の定数と先頭から1対1に対応する。①ではAに1.0，Bに2.5が初期値として設定される。

② DIMENSION　K (5)

DATA　K/5 * 0/

配列名を指定すると，すべての配列要素を指定したことになる。また，定数の繰返しの指定ができる。②は次のDATA文と同義。

DATA　K (1), K (2), K (3), K (4), K (5)/0, 0, 0, 0, 0/

③ DIMENSION　X(100)

DATA (X(I), I=1, 20) /20 * 1.0/

DO形並びを使用すると，配列の一部分に初期値を設定できる。③ではX (1)〜X (20)に1.0を設定している。

代入文

▶算術代入文

$v=e$
v は整数型，実数型の変数名または配列要素名。e は算術式。

機能►►式 e の値が v に代入される。式 e の値の型と v の型が異なる場合は，v の型に合わされてから代入される。

▶論理代入文

$v=e$
v は論理型の変数名または配列要素名。e は論理式。

機能►►式 e の値（真か偽）が v に代入される。

▶文番号代入文

ASSIGN s TO i
s は文番号，i は整数型の変数名。

機能►►文番号 s が変数 i に割り当てられる。文番号は，プログラム単位内の実行文または FORMAT 文の文番号であること。変数 i は割当て形 GO TO 文，または入出力文中の書式識別子で利用する。

▶文字代入文

$v=e$
v は文字型の変数名，配列要素名または部分列名。e は文字式。

機能►►式 e の値が v に代入される。

「v の長さ>e の長さ」の場合，e は左詰めで v に代入され，右側には空白が埋められる。「v の長さ<e の長さ」の場合，e の長い分が右側から切り捨てられ v に代入される。

制御文

▶単純 GO TO 文

GO TO *s*
s はプログラム単位内の実行文の文番号。

機能►►この文の実行により，文番号sをもつ文が次に実行される。

▶計算形 GO TO 文

GO TO (*s* [, *s*] …) [,] *i*
i は整数式，*s* はプログラム単位内の実行文の文番号。

機能►►この文の実行により，式 *i* が評価され，文番号の並び中の *i* 番目の文番号をもつ文が次に実行される。

①GO TO (10, 20) N

N が 1 の場合文番号10に，2 の場合文番号20へ移る。それ以外の場合は，この文の次の文へ実行が移る。

▶割当て形 GO TO 文

GO TO *i* [[,] (*s* [, *s*] …)]
i は整数型の変数名，*s* はプログラム単位中の実行文の文番号。

機能►►この文の実行により，変数 *i* に割り当てられている文番号の文が次に実行される。*i* に割り当てうる文番号を後ろに括弧を付けて並べてもよい。

```
① ASSIGN 100 TO I
  GO TO I
```

①の GO TO 文が実行されると，次は文番号100へ移る。

▶算術 IF 文

IF（e）s_1，s_2，s_3
e は整数式または実数式。s_1，s_2，s_3はプログラム単位内の実行文の文番号。

機能►►e＜0ならば次に文番号s_1の文が実行される。同様にe＝0ならばs_2，e＞0ならばs_3の文が実行される。

①IF (M+N)　10, 20, 30

M+Nの値が負ならば文番号10，0ならば文番号20，正ならば文番号30の文へ移る。

▶論理 IF 文

IF（e）st
e は論理式，st はDO文，ブロックIF文，ELSE IF文，ELSE文，END IF文，END文およびほかの論理IF文のいずれでもない実行文。

機能►►eの値が真ならば文 st が実行される。eの値が偽ならば文 st は実行されず，次の文へ移る。

①IF (N.GT.10)　X=0

Nの値が10より大きい場合だけXを0にする。

②IF (M+N.EQ.0)　GO　TO　10

M+Nが0ならば文番号10の文へ移る。M+Nが0でなければこの論理IF文の次の文へ移る。

▶ブロックIF文，ELSE文，END IF文

```
IF (e) THEN      ……ブロック IF 文
  [   H1   ]
ELSE             ……ELSE 文
  [   H2   ]
END IF           ……END IF 文
```

e は論理式。H_1，H_2には複数の文が指定でき，H_1を IF ブロック，H_2を ELSE ブロックという。

機能►► *e* の値が真ならばIFブロック，偽ならばELSEブロックが実行される。

①
```
IF (A.GT.B)  THEN
   X=A
  ELSE
   X=0
  END  IF
```

A>B ならば X=A，A≦B ならば X=0 が実行される。

②
```
IF (P.EQ.0)  THEN
   A=0
   B=0
  END  IF
```

P=0 のときだけ A=0, B=0 を実行する。P≠0 のときは何もしない。この場合も END IF 文は必ず指定すること。

③
```
IF (P.NE.0)  THEN
  ELSE
   A=0
   B=0
  END  IF
```

IFブロックは空でもよい。③ではP=0のときだけA=0, B=0を実行する。

```
④IF (P.EQ.0)  THEN
    IF (A.GT.B)  THEN
      X=A
    ELSE
      X=0
    END  IF
  END  IF
```

IFブロックやELSEブロックに，さらにブロックIF文を指定することができる。④のような場合，ELSE文は1番近いブロックIF文に対応する。よって，P=0かつA>Bの場合にはX=A，P=0かつA≦Bの場合にはX=0が実行される。

▶ELSE IF文

```
IF (e1) THEN
[ H1 ]
ELSE IF (e2) THEN
[ H2 ]
ELSE IF (e3) THEN
[ H3 ]
  ⋮
ELSE IF (en) THEN
[ Hn ]
ELSE
[ Hn+1 ]
END IF
```

e_1，e_2，…，e_nは論理式。H_1，H_2，…，H_{n+1}には複数の文が指定でき，H_2，H_3，…，H_nをELSE IFブロックという。

機能►►

e_1が真ならば	H_1	が実行される。
e_1が偽，e_2が真ならば	H_2	
e_1，e_2が偽，e_3が真ならば	H_3	
⋮	⋮	
e_1，e_2，…，e_{n-1}が偽，e_nが真ならば	H_n	
e_1，e_2，…，e_nすべてが偽ならば	H_{n+1}	

①
```
IF (A.GT.10)  THEN
  N=3
ELSE  IF (A.GT.5)  THEN
  N=2
ELSE
  N=1
END  IF
```

10＜Aのとき N=3，5＜A≦10のとき N=2，A≦5のとき N=1が実行される。

▶DO文

DO s [,] $i=e_1$，e_2 [，e_3]
sは実行文の文番号。iはDO変数といい，整数型，実数型の変数名。e_1，e_2，e_3はそれぞれ整数型，実数型の式。

機能►►繰返しを指定するのに用いる。DO文を使った繰返し処理の形は次のとおり。

```
DO   s   i=e1, e2, e3
s  [    H    ] } DOループ
```

処理Hの部分をDOループ，文番号sの文をDOループの端末文という。

e_1，e_2，e_3の値をそれぞれm_1，m_2，m_3とすると，DOループの繰返し数は次の式で定められる。ただし，e_3が省略されるとm_3は1となる。

MAX (INT (($m_2-m_1+m_3$)/m_3), 0)

このm_1，m_2，m_3をそれぞれ**初期値パラメタ**，**終値パラメタ**，**増分パラメタ**という。

DOループの端末文には次の文は指定できない。単純GO TO文，割当て形GO TO文，算術IF文，ブロックIF文，ELSE IF文，ELSE文，END IF文，RETURN文，STOP文，END文，DO文。

①
```
      DO 10 I=1, N
10    WRITE (6, *) I, I**2
```

$m_1>m_2$かつ$m_3>0$の場合と$m_1<m_2$かつ$m_3<0$の場合は繰返し数は0になる。①ではN＜1の場合，WRITE文は1回も実行されない。N≧1の場合は，DO変数Iの値が1からNまで変化しながらN回WRITE文が実行される。

②
```
      DO 10 J=1, N
        DO 20 I=1, M
          A (I, J)=0.0
20      CONTINUE
10    CONTINUE
```

DOループ中にさらに別のDOループを含めることができる。②では配列AのM行N列分の要素をすべて0にしている。この場合，次のように端末文を共有することができる。

```
      DO 10 J=1, N
      DO 10 I=1, M
        A (I, J)=0.0
10    CONTINUE
```

```
③       DO  100  I=N, 1, -1
          IF (K (I) . EQ. 0)    GO  TO  200
  100   CONTINUE
  200   WRITE (6, *)  I
```

DOループ中から，その範囲外へGO TO文などで制御を移した場合，DO変数はそのときの値を保持している。また，途中で飛び出すことなく繰返し処理を終えた場合，DO変数の値は「終値パラメタ＋増分パラメタ」の値になっている。

③ではK (N)，K (N−1)…と配列要素を順に調べていって，最初に現れた0の要素の添字式を出力している。0の要素がない場合は，繰返し処理を終了したときにIの値は1−1=0となっている。

```
④     DO   5   X=0.0, 10.0, 0.5
    5  WRITE (6, *)  X, X ** 2
```

DO変数に実数型のものも使用できる。④ではDO変数Xの値は0.0，0.5，1.0，1.5…と変化する。

▶ CONTINUE文

CONTINUE

機能►►この文の実行は何の効果ももたない。DOループの端末文によく使用される。

▶ STOP文

STOP [*n*]
*n*は文字定数または5けた以内の数字列。

機能►►実行が終了する。終了時に文字定数または数字列が参照可能となる。

▶ PAUSE 文

PAUSE [*n*]
n は文字定数または5けた以内の数字列。

機能►►実行が一時中断する。実行の一時中断時に文字定数または数字列が参照可能となる。

▶ END 文

END

機能►►プログラム単位の最後を示す。主プログラム内で実行すると，実行は終了する。

入出力文

▶ READ 文，WRITE 文

READ（*cilist*）[*iolist*] WRITE（*cilist*）[*iolist*]	
cilist を制御情報並びといい，次の項目を指定できる。	
[UNIT＝] *u*	[FMT＝] *f*
REC＝*rn*	IOSTAT＝*ios*
ERR＝*s*	END＝*s*
iolist を入出力並びといい，変数名，配列要素名，部分列名，配列名，DO 形並びを指定できる。WRITE 文ではさらに任意の式が指定できる。	

機能►► READ 文は入力並びに対しデータを入力する。WRITE 文は出力並びの値を出力する。

制御情報並びの各項目の意味は次のとおり。

(1) [UNIT＝] *u*

装置指定子といい，どのファイルに対し入出力を行うか，*u* に整数値を指定することで識別する。*u* を装置識別子という。

(2) [FMT＝] *f*

書式指定子という。これを含むものを**書式付き入出力**，含まないものを**書式なし入出力**という。書式付き入出力の場合，*f* の内容に従って変換を行い入出力を実行する。*f* を**書式識別子**といい，これに指定できるものは次のとおり。

- FORMAT 文の文番号
- FORMAT 文の文番号が割り当てられている整数型の変数名
- 文字配列名
- 任意の文字式

●星印(＊)

(3) REC＝*rn*

記録指定子という。これを含むものを**直接探査入出力**，含まないものを**順番探査入出力**という。直接探査の場合，何番目の記録に対し入出力を行うかをこの *rn* で指定する。

(4) IOSTAT＝*ios*

入出力状態指定子といい，誤り条件またはファイル終了条件が検出された場合，ゼロ以外の値が *ios* に指定した整数型の変数(配列要素）に返される。

(5) ERR＝*s*

誤り指定子といい，誤り条件を検出した場合，*s* に指定した文番号の文が次に実行される。

(6) END＝*s*

ファイル終了指定子といい，ファイル終了条件を検出した場合，*s* に指定した文番号の文が次に実行される。

① READ (5, 1, END＝90)　A, B

装置指定子の UNIT＝，書式指定子の FMT＝が省略された場合，*u* と *f* は制御情報並びの１番目と２番目に指定しなければならない。

①では５番のファイルからデータを変数 A，B に入力する。書式は文番号１の FORMAT 文に従い，ファイルの終りならば文番号90に飛ぶ。

② WRITE (10, REC＝5)　X

書式指定子がない場合，書式による変換は行われない。②は直接探査の出力で，５番目の記録に X の値を内部形式のまま出力する。

③ CH＝'(F 10.0, I 10)'
　READ (5, CH)　P, N

書式指定子に文字式または文字配列名が指定された場合，その内容の書式に従って入出力が行われる。③は次と同じ。

```
    READ (5, 1)  P, N
  1  FORMAT (F 10.0, I 10)
④ WRITE (6, *) P, N
```

書式指定子に星印（*）が指定されると，**並びによる入出力**が行われる。入力ではデータはけた数を気にせず，区切りをコンマか空白にして作っておけばよい。出力は処理系が適当に空白などで間を区切って行われる。

▶ OPEN文，CLOSE文

OPEN（*olist*） **CLOSE（*cllist*）**	
olist には次の項目が指定できる。	
[UNIT=] *u*	IOSTAT=*ios*
ERR=*s*	FILE=*fin*
STATUS=*sta*	ACCESS=*acc*
FORM=*fm*	RECL=*rl*
BLANK=*blnk*	
cllist には次の項目が指定できる。	
[UNIT=] *u*	IOSTAT=*ios*
ERR=*s*	STATUS=*sta*

機能 ▶▶ OPEN 文はファイルと装置の接続を行う。*olist* の各項目の意味は次のとおり。ただし，p.25にあげた制御情報並びに含まれるものは省く。

(1) FILE=*fin*

fin にはファイルの名前を文字式として指定する。

(2) STATUS=*sta*

ファイルの状態を指定する。*sta* が 'OLD' であれば，ファイルはすでにあるとみなされる。*sta* が 'NEW' の場合，ファイルは生成される。また，'SCRATCH' ではファイルは使用後消去される。

'UNKNOWN'では状態は処理系に依存する。

(3) ACCESS＝*acc*

ファイルの探査法を指定する。*acc* が 'SEQUENTIAL'であれば順番探査，'DIRECT'であれば直接探査となる。

(4) FORM＝*fm*

書式のありなしを指定する。*fm* が 'FORMATTED'であれば書式付き，'UNFORMATTED'ならば書式なしで入出力が行われる。

(5) RECL＝*rl*

直接探査の場合だけ必要。*rl* には記録の長さを整数式で指定。

(6) BLANK＝*blnk*

書式付き入出力において，記録に含まれる空白について指定する。*blnk* が 'NULL'であれば，入力欄中の空白はすべて無視される。ただし，空白の欄はゼロの値をもつものとみなされる。'ZERO'ならば，先行する空白はすべてゼロとみなされる。

CLOSE 文はファイルと装置の接続を終了させる。*clist* の各項目の意味は，*cilist* および *olist* の項目に含まれるので説明を省く。

```
①   OPEN (10, ACCESS= 'SEQUENTIAL',
  *  FORM= 'FORMATTED')
```

10番のファイルについての OPEN 文。順番探査でかつ書式付き入出力を行う。

```
②   OPEN (20, ACCESS= 'DIRECT',
  *  FORM= 'UNFORMATTED',
  *  RECL=20)
```

20番のファイルについての OPEN 文。直接探査でかつ書式なしで入出力を行う。なお，記録の長さは20となっている。この OPEN 文に対応する CLOSE 文は次のように指定すればよい。

```
CLOSE (20)
```

▶ BACKSPACE文，ENDFILE文，REWIND文

BACKSPACE *u* ENDFILE *u* REWIND *u*
*u*には装置識別子を指定する。

機能►►BACKSPACE文の実行により，現在の記録の直前の記録に位置付けられる。

ENDFILE文の実行により，現在の記録の後にファイル終了記録が書かれる。

REWIND文の実行により，ファイルの始点に位置付けられる。

▶書式仕様

([*flist*])
書式仕様は次の方法で与える。 ● FORMAT文 ● 文字配列，文字変数またはほかの文字式の値

機能►►書式付き入出力において，ファイルの記録中のデータと内部形式との変換を行う。

flist 中に指定する編集記述子には，繰返しの指定ができる反復可能編集記述子とできない反復不能編集記述子がある。それぞれの編集記述子の種類をあげる。

(1)反復可能編集記述子

I*w*	I*w*. *m*	
F*w*. *d*	E*w*. *d*	E*w*. *d*E*e*
G*w*. *d*	G*w*. *d*E*e*	
L*w*		
A	A*w*	

(2)反復不能編集記述子

'$h_1h_2\cdots h_n$'　　$n\mathrm{H}h_1h_2\cdots h_n$

Tc　　TLc　　TRc

nX

/

:

S　　SP　　SS

kP

BN　　BZ

反復可能編集記述子は，次のように繰返しの指定をし，まとめて書くことができる。

I5, I5, I5 → 3I5

編集記述子と入出力並び項目の対応は次のようになる。

● 反復可能編集記述子の数が入出力並び項目の数より多い場合，余った編集記述子は無視される

```
  WRITE (6, 1)  A, N
1 FORMAT ('1', F10.1, I10, F10.1)
```

（F10.1）無視される

● 反復可能編集記述子の数が入出力並び項目の数より少ない場合，書式仕様が繰り返される。この場合，右から2番目の右括弧に対応する左括弧に戻る

```
  READ(5, 1)  N, (X (I), I=1,N)
1 FORMAT(I10/(F10.1))
```

1枚目　2枚目　3枚目

次に，それぞれの編集記述子の機能について述べる。

① I*w*　I*w*. *m*

整数型の編集に用いる。*w* は全けた数。出力では I*w*. *m* の場合少なくとも *m* けたになるように，先頭に必要ならば 0 を付ける。

入力欄	編集記述子	入力された値
␣␣123	I5	123
␣−123	I5	−123
␣␣␣12	I5.3	12

出力される値	編集記述子	出力欄
123	I5	␣␣123
123	I5.4	␣0123

② F*w*. *d*　E*w*. *d*　E*w*. *d*E*e*

実数型の編集に用いる。*w* は全けた数。入力ではどれも同じ働きをし，*d* は小数点以下のけた数になる。ただし，入力データ中に小数点が含まれる場合はこちらが優先する。

出力では形式が異なる。F*w*. *d* は小数点以下 *d* けたで出力される。E*w*. *d* および E*w*. *d*E*e* は指数部付きで，次のような形で出力される。仮数部の小数点以下が *d* けたとなる。なお，E*w*. *d*E*e* の *e* は指数部のけた数となる。

$\pm 0 . x_1 x_2 \cdots x_d \; exp$

入力欄	編集記述子	入力された値
␣␣301	F5.1	30.1
␣␣−556	E6.3	−0.556

出力される値	編集記述子	出力欄
0.1234×10^{2}	F7.1	␣␣␣12.3
0.9876×10^{2}	E10.3	␣0.988E+02
0.357×10^{-5}	E12.3E3	␣␣0.357E−005

③ G*w*. *d*　G*w*. *d*E*e*

実数型の編集に用いる。入力では F*w*. *d* と同じ働きをする。出力

では出力される値の絶対値によって表現が異なる。データの絶対値をNとすると，N<0.1またはN≧10＊＊dのときは，E*w*. *d* またはE*w*. *d*E*e* と同じ。それ以外のときは次のようになる。ただし，*n* はG*w*. *d* のとき4，G*w*. *d*E*e* のとき *e*+2とする。

データの絶対値	実行される編集
0.1≦N<1	F(*w*−*n*). *d*, *n* ('␣')
1≦N<10	F(*w*−*n*). (*d*−1), *n* ('␣')
⋮	⋮
10＊＊(d−2)≦N<10＊＊(d−1)	F(*w*−*n*). 1, *n* ('␣')
10＊＊(d−1)≦N<10＊＊d	F(*w*−*n*). 0, *n* ('␣')

④ L*w*

論理型の編集に用いる。*w* は全けた数。入力データとしてはTやFの文字の前に空白，コンマがあってもよい。また，TやFに続いて任意の文字があってよい。

入力欄	編集記述子	入力された値
␣␣T␣␣	L6	真
␣.FALSE.	L8	偽

出力される値	編集記述子	出力欄
真	L5	␣␣␣␣T
偽	L3	␣␣F

⑤ A　A*w*

文字型の編集。*w* は全けた数。*w* を指定しないと，入出力項目の長さと同じ文字数が仮定される。

入力欄がABCDEFGHIであるとする。

編集記述子	入力された値(入力先は長さ5とする)
A3	ABC␣␣
A7	CDEFG
A	ABCDE

出力される値はABCDEFGHI（長さ9）とする。

編集記述子	出力欄
A	ABCDEFGHI
A5	ABCDE
A12	␣␣␣ABCDEFGHI

⑥ '$h_1h_2\cdots h_n$'　　nH$h_1h_2\cdots h_n$

文字列 $h_1h_2\cdots h_n$の出力に用いる。

編集記述子	出力欄
'FORTRAN'	FORTRAN
5HJOHOU	JOHOU

⑦ Tc　　TLc　　TRc

記録の任意の文字への位置付けに用いる。Tc は記録の先頭から c 文字目に位置付ける。TLc は現在の位置から左へ c 文字分ずらす。TRc は現在の位置から右へ c 文字分ずらす。

```
   READ (5, 1)  A, B, C
 1 FORMAT (T10, F5.1, T1, F5.1, T20, F5.1)
```

この例では10～15けた目を変数A，1～5けた目をB，20～25けた目をCに入力する。

⑧ nX

記録中での現在の文字位置から n 文字右にずらす。出力では空白が出力され，入力では n けた読み飛ばすことができる。

⑨ /

記録の区切りを示す。

```
   READ (5, 1)  A, B, C
 1 FORMAT (/F5.1//F5.1//F5.1)
```

2枚目のカードから変数Aに，4枚目からBに，6枚目からCにデータを入力する。

```
   WRITE(6, 2)  A, B, C
```

2 FORMAT('1','A=', F10.1/'', 2F10.1)

1行目にAの値，2行目にBとCの値を出力する。

⑩：

入出力並びに項目が残っていない場合，書式制御を終了させる。

WRITE (6, 1) X

1 FORMAT('0', 'X=', F10.1, :, '␣Y=', F10.1)

X=の文字と変数Xの値が出力される。次の：がないとY=の文字まで出力されてしまう。

⑪S　SP　SS

数値の出力で正符号+を付けるか付けないかの指定に用いる。

SPが現れると以降は正符号が付く。SSが現れると以降正符号は付かない。Sが現れると処理系の定めた形に戻る。

WRITE (6, 1) 123, 45

1 FORMAT ('0', SP, I5, 2X, SS, I5)

最初の出力項目123は正符号が付き+123と出力される。45は正符号は付かずに出力される。

⑫*k*P

けた移動数を指定するために用いる。

入力では入力欄中に指数部のないデータに対してだけ有効となり，10 ＊＊(−*k*)倍されて入力される。

入力欄	編集記述子	入力された値
␣␣12.3	1PF6.1	1.23
␣−456	−2PF5.0	−45600.0

出力ではE*w*. *d*およびE*w*. *d*E*e*に対して，仮数部に10 ＊＊ *k*が掛けられ指数は*k*だけ減らされる。

出力される値	編集記述子	出力欄
0.1234×10^{2}	1PE10.3	␣1.234E+01
0.45678×10^{-2}	3PE10.1	␣456.8E−05

けた移動数は各入出力文の開始時点では0である。しかし，*k*P で一度変更すると以降そのけた移動数 *k* が有効になる。

⑬BN　　BZ

BNを指定すると，それに続く数値の入力は，データ中の空白がすべて無視される。

BZ を指定すると，逆にデータ中の空白は，すべて0(ゼロ)として扱われる。

組込み関数

▶組込み関数の引用

fun (*a* [, *a*] …)
fun は組込み関数の個別名または総称名。*a* は引数。

機能►►よく使用される関数は，組込み関数としてあらかじめ用意されている。総称名を使用した場合，関数の型つまり結果の型は引数の型に依存する。

① N＝IABS (K)

N＝ABS (K)

絶対値を求める関数では，引数が整数型の場合，個別名 IABS を使う。また，総称名は ABS なので，下のように指定してもよい。

②主プログラム側

```
INTRINSIC  ALOG
   ⋮
CALL  SUB (ALOG, X, Y)
```

サブルーチン側

```
SUBROUTINE  SUB  (F, A, B)
   ⋮
W＝F (A＋B)
   ⋮
```

組込み関数の個別名を実引数に指定し，サブルーチンや関数副プログラムに渡すことができる。この場合，INTRINSIC 文でその関数名を指定すること。

②の例ではサブルーチン SUB の仮引数 F と ALOG が対応する。よって，サブルーチン内の W＝F(A＋B)はこの場合 W＝ALOG (A＋B)となる。

▶組込み関数一覧（一部分）

組込み関数	定義	引数の個数	総称名	個別名	引数の型	関数の型
型変換	整数型への変換 int(a)	1	INT	INT IFIX	実数型 実数型	整数型 整数型
	実数型への変換	1	REAL	REAL FLOAT	整数型 整数型	実数型 実数型
	整数型への変換	1		ICHAR	文字型	整数型
	文字型への変換	1		CHAR	整数型	文字型
切捨て	int(a)	1	AINT	AINT	実数型	実数型
四捨五入	$a \geqq 0$ならば int($a+0.5$) $a<0$ならば int($a-0.5$)	1	ANINT	ANINT	実数型	実数型
四捨五入整数化	$a \geqq 0$ならば int($a+0.5$) $a<0$ならば int($a-0.5$)	1	NINT	NINT	実数型	整数型
絶対値	$\lvert a\rvert$	1	ABS	IABS ABS	整数型 実数型	整数型 実数型
剰余	$a_1-\mathrm{int}(a_1/a_2)*a_2$	2	MOD	MOD AMOD	整数型 実数型	整数型 実数型
符号の付替え	$a_2 \geqq 0$ならば $\lvert a_1\rvert$ $a_2<0$ならば $-\lvert a_1\rvert$	2	SIGN	ISIGN SIGN	整数型 実数型	整数型 実数型
超過分	$a_1>a_2$ならば a_1-a_2 $a_1 \leqq a_2$ならば 0	2	DIM	IDIM DIM	整数型 実数型	整数型 実数型
最大値	max($a_1, a_2, \cdots$)	≧2	MAX	MAX0 AMAX1	整数型 実数型	整数型 実数型
				AMAX0 MAX1	整数型 実数型	実数型 整数型
最小値	min($a_1, a_2, \cdots$)	≧2	MIN	MIN0	整数型	整数型

				AMIN1	実数型	実数型
				AMIN0 MIN1	整数型 実数型	実数型 整数型
文字長	文字要素の長さ	1		LEN	文字型	整数型
部分列の位置	文字列 a_1 のなかの部分列 a_2 の位置	2		INDEX	文字型	整数型
平方根	$(a)^{\frac{1}{2}}$	1	SQRT	SQRT	実数型	実数型
指　数	$e**a$	1	EXP	EXP	実数型	実数型
自然対数	$\log_e(a)$	1	LOG	ALOG	実数型	実数型
常用対数	$\log_{10}(a)$	1	LOG10	ALOG10	実数型	実数型
正　弦	$\sin(a)$	1	SIN	SIN	実数型	実数型
余　弦	$\cos(a)$	1	COS	COS	実数型	実数型
正　接	$\tan(a)$	1	TAN	TAN	実数型	実数型
逆正弦	$\arcsin(a)$	1	ASIN	ASIN	実数型	実数型
逆余弦	$\arccos(a)$	1	ACOS	ACOS	実数型	実数型
逆正接	$\arctan(a)$	1	ATAN	ATAN	実数型	実数型
	$\arctan(a_1/a_2)$	2	ATAN2	ATAN2	実数型	実数型
双曲線正弦	$\sinh(a)$	1	SINH	SINH	実数型	実数型
双曲線余弦	$\cosh(a)$	1	COSH	COSH	実数型	実数型
双曲線正接	$\tanh(a)$	1	TANH	TANH	実数型	実数型
文字列の大小比較	$a_1 \geqq a_2$	2		LGE	文字型	論理型
	$a_1 > a_2$	2		LGT	文字型	論理型
	$a_1 \leqq a_2$	2		LLE	文字型	論理型
	$a_1 < a_2$	2		LLT	文字型	論理型

文関数

▶文関数定義文

fun ([*d* [, *d*] …])=*e*
fun は文関数名，*d* は文関数の仮引数，*e* は式とする。

機能►►文関数を算術式，論理式あるいは文字式の形で定義する。その文関数は文関数定義文のあるプログラム単位内で引用できる。

①F (X)=X ＊＊ 2

⋮

Y=F (A+B)

文関数の引用は関数名の後に括弧で囲んで実引数を指定すればよい。①では変数 Y には(A+B)＊＊ 2 が代入される。

②INTEGER G

G (X, N)=SQRT (X ＊ N)

文関数の型つまり結果の型は，文関数名の型で決まる。②では文関数名 G は整数型なので，文関数定義文の式の値が整数化されたものが結果となる。

主プログラム

▶ PROGRAM 文

PROGRAM *pgm*
pgm の付け方の規則は，変数名の場合と同じ。ただし，型はない。

機能►► PROGRAM 文は主プログラムの先頭に指定し，その主プログラムに名前を付ける。

副プログラム

▶ FUNCTION 文

[*typ*] FUNCTION *fun* ([*d* [, *d*] …])
typ は INTEGER, REAL, LOGICAL, CHARACTER[∗ *len*]のいずれか。*fun* は外部関数名。*d* は変数名，配列名または仮手続き名とし，仮引数という。

機能►►関数副プログラムの先頭に必ず指定する。関数の型は関数名 *fun* の型で決まる。FUNCTION 文の先頭で型を指定することもできる。

```
①REAL  FUNCTION  F(X)
  IF (X.GT.0)  THEN
     F=SQRT(X)
   ELSE
     F=0
   END  IF
  RETURN
  END
```

関数の値は関数名に代入された値となる。RETURN 文が実行されると，呼出し元のプログラム単位に戻る。最後は END 文を指定すること。

関数副プログラムの引用は関数名の後に括弧で囲んで実引数を指定する。

①の関数副プログラム F は次のような関数を定義している。

$$f(x)=\begin{cases}\sqrt{x} & (x>0)\\ 0 & (x\leqq 0)\end{cases}$$

$f\ (a+b)$ を求めたければ次のように引用する。$A+B$ を実引数と

いい，実引数には式が指定できる。仮引数と実引数は先頭から１対１に対応する。

$Y=F\ (A+B)$

関数副プログラムは１つのプログラム単位なので，そのなかで使用する変数名，配列名，文番号などはすべてほかのプログラム単位と独立になる。

```
②    FUNCTION SUM (A, N)
      DIMENSION A (0:N)
      DO 10 I=0, N
   10 SUM=SUM+A (I)
      RETURN
      END
```

仮配列（仮引数の配列）の宣言に，仮引数または共通ブロックの要素を使用できる。このような配列を**整合配列**という。

②では仮配列Ａの宣言に仮引数Ｎを利用している。もしＮに10渡されれば，配列ＡはＡ(0)～Ａ(10)の11個の要素をもつものと宣言される。

```
③ DIMENSION A (0: *)
```

整合配列の宣言で仮引数を指定したところに＊を指定すると，**擬寸法仮配列**になる。このときの仮配列Ａの大きさは，対応する実配列（実引数の配列）と同じになる。

```
④ REAL FUNCTION F (X, SUB)
          ⋮
   CALL SUB (X)
          ⋮
   END
```

仮引数としてほかの副プログラムの名前を受け取ることができる。これを**仮手続き**という。

ただし，呼出し元では，実引数に指定した副プログラムの名前をEXTERNAL文で宣言しておく必要がある。

④の例では仮引数SUBにサブルーチン名を受け取っている。呼出し元のプログラムでサブルーチンMULTを渡す場合，次のようになる。

```
EXTERNAL  MULT
Q=F (A, MULT)
```

▶ SUBROUTINE文

SUBROUTINE *sub* [([*d* [, *d*] …])]
sub はサブルーチン名。*d* は変数名，配列名または仮手続き名とし，仮引数という。

機能▶▶サブルーチン副プログラムの先頭に必ず指定する。

①
```
SUBROUTINE  SUB (X, Y)
  IF (X. GT. 0)  THEN
     Y=SQRT (X)
   ELSE
     Y=0
   END  IF
  RETURN
  END
```

サブルーチン副プログラムの引用はCALL文で行う。

```
CALL  SUB (A+B, C)
```

このCALL文で変数Cにサブルーチンで求めた結果が返される。

サブルーチン副プログラムも，関数副プログラム同様，整合配列と擬寸法仮配列が使用できる。

▶ BLOCK DATA文

BLOCK DATA [*sub*]
sub は初期値設定副プログラムの名前。

機能►►初期値設定副プログラムの先頭に必ず指定する。

①
```
BLOCK  DATA
COMMON  /BLK/A, B, C
DATA  A, B, C/1.0, 2.0, 3.0/
END
```

初期値設定副プログラムでは，DATA文により名前付き共通ブロックの各要素に初期値を設定できる。指定できる文は，IMPLICIT文，PARAMETER文，DIMENSION文，COMMON文，EQUIVALENCE文，DATA文，END文および型宣言文だけである（注釈行はあってもよい）。

共通ブロック

▶ COMMON 文

COMMON [/*cb*/]　*nlist* [[,]/ [*cb*]/*nlist*] …
cb は共通ブロック名。*nlist* は変数名，配列名，配列宣言子の並び。

機能►►共通ブロックの宣言をする。共通ブロック中のデータは，異なるプログラム単位で引数を使わず受け渡しができる。

共通ブロック名のないものを**無名共通ブロック**という。

①COMMON　A, B/BLK1/X, Y (10)

最初が無名共通ブロックの場合，2つの斜線はなくてもよい。配列宣言子を指定して，配列の宣言も行える。

①では要素A，Bの無名共通ブロック，要素X，Y(配列)の共通ブロックBLK1の宣言である。

② 主プログラム

```
COMMON  /BK/P, Q
   :
CALL  SUB
   :
```

サブルーチン副プログラム

```
SUBROUTINE  SUB
COMMON  /BK/X, Y
   :
Y=X＊＊2
RETURN
END
```

共通ブロックを介して，プログラム単位間でデータのやりとりができる。この場合，各プログラム単位にCOMMON文が必要である。要素の名前は独立に付けられるが，対応は先頭から1対1に付けられる。

和文索引 (50音順)

は－ほ

ま－も

ら－ろ

わ

欧文・記号索引

COBOL
基本文法ハンドブック

日本工学院八王子専門学校情報処理科
太田宗雄/中山二夫

▶このCOBOL基本文法ハンドブックは，COBOLの初心者から中級者までを対象に，プログラミング上の規則や基本命令の使い方などを解説したものです。

▶文法事項は，日本工業規格 JIS COBOL C6205-1980に準拠しています。

▶本文中の一般形式に表示された記号類は，次の意味を表します。

① [ ] のなかは，必要に応じて書いても書かなくてもよい

② { } のなかは，このうちのどれか1つを選んで書かなくてはならない

③ [ ] か { } のあとに…がある場合は，かっこ内の内容を繰り返し書いてもよい

④＿＿＿（下線）のある予約語は，その形式を使う場合に必ず書かなければならない。下線のない予約語は，書いても書かなくてもよい

⑤＋，－，<，>，＝は，下線がなくても，その形式を使う場合は必ず書かなければならない

COBOL
基本文法ハンドブック――目次

COBOLとは

▶COBOLの歴史

1959年5月，アメリカ国防省で，事務処理用の共通性のある言語の開発を検討する会合が開かれた。これにコンピュータメーカー，ユーザとが協賛し，COBOL(COmmon Business Oriented Language 事務用共通言語)を開発することになった。この作業の中心になったのが，CODASYL (COnference on DAta SYstems Languages データシステムズ言語協議会) である。

1960年4月に第1回仕様書が発表され，COBOLが誕生した。これがCOBOL-60である。以後，ハードウェアの機能が向上するに従って，改訂増補されて今日に至っている。

日本においても，COBOLの標準化作業が日本工業規格 (JIS) の下で行われるようになった。その最初の規格は，1972年8月に制定されたJIS COBOL-1972である。その後，JIS COBOL-1980が制定されて現在に至っているが，間もなく3回目のJIS COBOLが発表されることになっている。

▶COBOLの特徴

COBOLが開発されたときの主な留意点は，互換性，記述性，平易性であり，それがそのままCOBOLの特徴となっている。

①**互換性**：現在使用しているコンピュータから，より上位の機種に設置移行する場合，プログラムを作成する費用，労力，時間を低減できる。

②**記述性**：日常語（英語）に近い形式で記述されるので，プログラム自体が文章としての機能をもち，これによって，担当者間の意思の疎通を図ることができる。

③**平易性**：ハードウェアの知識を熟知していなくてもCOBOLの文法が理解でき，簡単にプログラムが作成できる。

プログラムの構成

プログラムは，コンピュータに対する指示書である。したがって，そのプログラムのなかには，“どのようなデータをどのように計算処理し，どの装置にどんな形式で結果を書き出すか”，という情報を含んでいなければならない。

COBOL のプログラムは，4つの独立した部（DIVISION）とよばれるもので構成されている。4つの部は次の順序で記述しなければならない。

▶見出し部（IDENTIFICATION DIVISION）

COBOL プログラムの「見出し」ともいうべきもので，プログラムの名前，作成者名，作成年月日などについて記述する。

▶環境部（ENVIRONMENT DIVISION）

COBOL プログラムが翻訳，実行される場合のコンピュータの機器構成，および，変換された実行用プログラムを実行する場合のデータファイルと入出力機器との関連などについて記述する。

▶データ部（DATA DIVISION）

処理の対象となるデータファイル，作業領域として定義されるデータ，および，プログラムで使用される定数などの構造，大きさ，種類などについて記述する。

▶手続き部（PROCEDURE DIVISION）

データファイルを読み，計算を行い，その結果を書き出す，という一連のデータ処理を行うのに必要な手続きについて記述する。

以上の4つの部で構成される COBOL プログラムは，COBOL で扱う文字を用いて記述する。COBOL で扱う文字は，語や定数などを作るために用いられる。文字の種類は，英字（A～Z），数字（0～9），および，その他の特殊文字やカナ文字なども含まれる。

COBOLのソースプログラム例

```
    IDENTIFICATION  DIVISION.
    PROGRAM-ID.          URIAGE.
    AUTHOR.              TOKYO TARO.
    ENVIRONMENT       DIVISION.
    CONFIGURATION  SECTION.
    SOURCE-COMPUTER.    COMP.
    OBJECT-COMPUTER.    COMP.
    INPUT-OUTPUT  SECTION.
    FILE-CONTROL.
        SELECT  URIAGE-F  ASSIGN TO  INP.
        SELECT  HYO-F     ASSIGN TO  OUT.
    DATA              DIVISION.
    FILE             SECTION.
    FD  URIAGE-F  LABEL  RECORD  STANDARD.
    01  URIAGE-R.
        02  TANKA           PIC 9(3).
        02  SURYO           PIC 9(3).
    FD  HYO-F     LABEL  RECORD  STANDARD.
    01  HYO-R.
        02  FILLER          PIC X(5).
        02  H-TANKA         PIC ZZ9.
        02  FILLER          PIC X(5).
        02  H-SURYO         PIC ZZ9.
        02  FILLER          PIC X(5).
        02  H-KINGAKU       PIC ZZZ,ZZ9.
    WORKING-STORAGE  SECTION.
    01  W-KINGAKU           PIC 9(6).
    01  MIDASHI             PIC X(28) VALUE IS
        "     ﾀﾝｶ   ｽｳﾘﾖｳ         ｷﾝｶﾞｸ".
    PROCEDURE         DIVISION.
    HAJIME.
        OPEN INPUT  URIAGE-F  OUTPUT  HYO-F.
        WRITE HYO-R  FROM MIDASHI AFTER ADVANCING PAGE.
        MOVE SPACE TO HYO-R.
    YOMU.
        READ  URIAGE-F  AT  END  GO TO  OWARI.
        COMPUTE  W-KINGAKU  =  TANKA  *  SURYO.
        MOVE  TANKA        TO  H-TANKA.
        MOVE  SURYO        TO  H-SURYO.
        MOVE  W-KINGAKU    TO  H-KINGAKU.
        WRITE  HYO-R  AFTER  ADVANCING  2  LINES.
        GO TO  YOMU.
    OWARI.
        CLOSE  URIAGE-F  HYO-F.
        STOP  RUN.
```

COBOLの語

COBOLの語は，30文字以内の文字列で構成される。文字列とは，COBOLのプログラムの最小の構成単位であり，予約語と利用者語がある。

▶ **予約語**

COBOLの文法上，あらかじめ用途の決められた語を予約語とよぶ。すなわち，COBOLコンパイラ（翻訳プログラム）によって，ある一定の意味をもつよう決められている語であるため，プログラマはこの予約語を利用者語（データ名，手続き名など）に用いることできない。

予約語は必要語，補助語，その他の予約語に分けられる。

①**必要語**：ある1つの文を書くときに，必ず書かなければならない語を必要語という。たとえば，

ADD DATA-A TO DATA-B.

という命令において，下線部分のADDとTOは，加算を行わせる場合には必ず書かなければならない必要語である。

②**補助語**：COBOLのプログラムを読みやすくするために，適時挿入される語を補助語という。補助語を書くか書かないかは，プログラマの任意である。たとえば，

IF DATA-C IS LESS THAN DATA-D ….

という命令において，下線部分のISとTHANは，補助語なので，省略しても文法上は正しく扱われる。

③**その他の予約語**：連結語のOF，IN，AND，OR，比較文字の<，>，=，表意定数のZERO，SPACEなどが含まれる。

▶ **利用者語**

利用者が決めるデータ名や手続き名は利用者語とよばれ，予約語でない30文字以内の語として作成される。

COBOLの定数

定数は，それを構成する文字そのもの，または，表意定数の指定によって決められる。定数の種類は，文字定数，数字定数，表意定数のいずれかである。

▶ 文字定数

文字定数とは，コンピュータが取り扱う文字の任意の文字列を，引用符（〝 〟）で囲んだものである。

実行用プログラムにおける文字定数の値は，引用符を外した文字列そのものである。たとえば，「COBOL プログラム」という文字列を値としたいときは，〝COBOL プログラム〟と書く。

▶ 数字定数

数字定数とは，0〜9 の数字，正負記号，実小数点からなる文字列である。数字定数を作るさいの主な規則は，次のとおりである。

①少なくとも 1 字の数字を含むこと。

②正負記号（+，−）は，先頭の文字位置に 1 個だけ書ける。

③実小数点は，右端の文字位置を除いて 1 個だけ書ける。

なお，数字定数の値は，数字定数内の文字列によって表される代数的な値そのものである。

▶ 表意定数

表意定数は，決められた予約語で指定する。表意定数の単数形と複数形は同じ意味に解釈されるので，どちらを使ってもよい。

ZERO ……………………………………値ゼロまたは何けたかのゼロを表す
SPACE ………………………………………………何けたかの空白を表す
HIGH-VALUE …………………………………何けたかの最大文字を表す
LOW-VALUE ……………………………………何けたかの最小文字を表す
ALL　定数 ………………………………定数文字列の何回かの反復を表す

見出し部

COBOL のプログラムは，必ず，見出し部(IDENTIFICATION DIVISION) から書かなければならない。利用者は，以下に示す形式の段落のなかに，プログラムの名前，作成者の名前，作成日などの情報を書くことができる。

段落の見出しを用いて，その段落に含まれている情報の見出しとする。段落の見出しは，A 領域（カラム 8～11）から書いて，その後に終止符と空白をつけたものである。

最初の段落として，必ず PROGRAM-ID (プログラム名) 段落を記述し，ここにプログラムの名前を書かなければならない。これ以外の段落の記入は任意であり，以下に示す見出し部の構成の順序で，この部に書くことができる。なお注記項には COBOL の文字なら何を書いてもよいが，必ず，ピリオドと空白で終わらなければならない。

▶見出し部の構成

〈一般形式〉

```
  IDENTIFICATION DIVISION.
  PROGRAM-ID.          プログラム名.
 [AUTHOR.              注記項.]
 [INSTALLATION.        注記項.]
 [DATE-WRITTEN.        注記項.]
 [DATE-COMPILED.       注記項.]
 [SECURITY.            注記項.]
```

▶段落の機能

段落の機能は，各段落の見出しがその意味を表している。PROGRAM-ID はプログラム名段落。AUTHOR は作成者名段落。INSTALLATION は設置場所段落。DATE-WRITTEN は作成日付け段落。DATE-COMPILED は翻訳日付け段落。SECURITY は機密段落。

環境部

環境部(ENVIRONMENT　DIVISION)には，構成節と入出力節があり，翻訳および実行時に使用するコンピュータの構成と入出力されるデータファイルについて記述する。

▶環境部の構成

```
〈一般形式〉
  ENVIRONMENT  DIVISION.
  CONFIGURATION  SECTION.
  SOURCE-COMPUTER.        翻訳用計算機記述項.
  OBJECT-COMPUTER.        実行用計算機記述項.
 [INPUT-OUTPUT  SECTION.
  FILE-CONTROL.           {ファイル管理記述項.} …]
```

▶構成節（CONFIGURATION　SECTION）

構成節には，翻訳用計算機と実行用計算機の特性を書く。

①翻訳用計算機段落（SOURCE-COMPUTER）

この段落では，COBOL の原始プログラム(source　program)を翻訳するコンピュータの名前を記述する。

②実行用計算機段落（OBJECT-COMPUTER）

この段落では，翻訳された目的プログラム(object　program)を実行するコンピュータの名前を記述する。

▶入出力節（INPUT-OUTPUT　SECTION）

入出力節のファイル管理段落(FILE-CONTROL)では，ファイルに名前をつけ，外部媒体を割り当てる。すなわち，システムが認識している名前（作成者語）に対して，プログラムが認識する名前（ファイル名）を割り当てる。

```
〈一般形式〉
  FILE-CONTROL.
      SELECT  ファイル名  ASSIGN  TO  作成者語.
```

データ部

データ部（DATA　DIVISION）は，実行用プログラムが行う入力処理，内部処理，出力処理で使用されるデータについて記述する。

データ部の主要な節は，ファイル節（FILE　SECTION）と作業場所節（WORKING-STORAGE　SECTION）である。ファイル節では入出力されるファイルデータについて定義し，作業場所節では定数データや中間データを定義する。

▶ データ部の構成

▶ ファイルとレコード

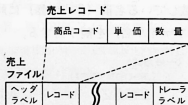
コンピュータでデータを処理する場合，その処理に必要なデータを用意しなければならない。たとえば，給与明細表を作成するための１人１人の給与データや，売上一覧表を作成するための１件１件の売上データなどである。このように１回の処理に必要な１人分のデータや１件分のデータのことをレコードとよぶ。

これらのレコードは，カード，磁気テープ，磁気ディスクなどに記録して保存される。このレコードを集めたものをファイルとよぶ。ファイルには，通常の本のように，表紙，裏表紙に相当するラベル(label)がつけられることが多い。

売上レコード

商品コード	単　価	数　量

売上ファイル

ヘッダラベル	レコード	≀≀	レコード	トレーララベル

ファイル節と作業場所節

入出力ファイルを用いる場合は，必ずファイル節を記述しなければならない。また，中間データや定数データを必要とするときは，作業場所節を記述する。

▶ファイル節

ファイル節は，ファイル記述項とそれにつづくいくつかのレコード記述項で構成される。

①ファイル節の構成

〈一般形式〉

FILE SECTION.

[ファイル記述項
{レコード記述項} …] …

②ファイル記述項：ファイル記述項は，ファイルの名前とファイルの物理的な構造（ラベルの有無など）を与える。

〈一般形式〉

FD ファイル名

LABEL {RECORD IS / RECORDS ARE} {STANDARD / OMITTED}.

③レコード記述項：レコード記述項は，レコードの階層構造に従ってデータ定義を行う。

▶作業場所節

作業場所節は，内部処理で発生する中間データや定数データの項目について記述する。この節で定義するデータ項目には，独立項目と集団項目（レコード）がある。

①作業場所節の構成

〈一般形式〉

WORKING-STORAGE SECTION.

[独立項目記述項] …

[レコード記述項] …

データ記述項

データ記述項は，ファイル節のレコード記述や作業場所節のデータ記述の内容を具体的に指定する。

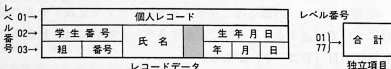
▶ データ記述項の構成

```
レベル番号 {データ名 } [{PICTURE} IS  picture文字列 ]
           {FILLER  }  {PIC    }
                     [VALUE IS  定数]
```

▶ データの階層とレベル番号

COBOL では，レコードデータを細分化して階層的に定義することができる。この場合，論理的にそれ以上細分化されない項目を基本項目とよび，基本項目または集団項目の集まりを集団項目とよぶ。また，階層の対応関係をコーディングするときはレベル番号を用いる。レベル番号には，01～49 や 77 などがある。

階層関係をもつレコードデータは 01～49 のレベル番号を用い，最上位の階層にはレベル番号 01 を，下位の階層にはそれより大きいレベル番号を与える。

階層関係をもたないデータ項目は独立項目とよばれ，作業場所節で定義される。この項目には，レベル番号 01 または77 が用いられる。

▶ データ名と FILLER

データ名とは，利用者が定義する語である。このデータ名は，手続き部で参照される名前となる。また，レコード中に直接参照され

ないデータ項目がある場合は，FILLER（無名項目）を用いる。

▶ PICTURE 句

PICTURE 句では，基本項目および独立項目のデータの種類や大きさを指定するために，PICTURE 文字列を用いる。

基本的な PICTURE 文字としてよく用いられるものに，9，X，S，V がある。9 は数字 1 けたを表し，X は任意の文字 1 けたを表す。また，S と V は 9 といっしょに用いられ，それぞれ，符号と想定小数点位置を表す。なお，S と V は項目の大きさに数えられない。

▶ VALUE 句

VALUE 句は，作業場所節のデータ項目に初期値を与える。

〔例 1〕レコードデータの定義

KOJIN-R						
GAKUSEI		SHIMEI		UMARE		
KUMI	BANGO			NEN	TUKI	NICHI
X(2)	X(3)	X(15)	X(10)	9(2)	9(2)	9(2)

```
01  KOJIN-R.
    02  GAKUSEI.
        03  KUMI     PIC  X(2).
        03  BANGO    PIC  X(3).
    02  SHIMEI       PIC  X(15).
    02  FILLER       PIC  X(10).
    02  UMARE.
        03  NEN      PIC  9(2).
        03  TUKI     PIC  9(2).
        03  NICHI    PIC  9(2).
```

〔例 2〕独立項目の定義

	MOJI		
	X(3)		
（初期値）	A	B	C

	ATAI			
	9(3)V9			
（初期値）	1	2	3	4

```
01  MOJI  PIC  X(3)  VALUE  "ABC".
01  ATAI  PIC  9(3)V9  VALUE  123.4.
```

編集用PICTURE文字

編集用PICTURE文字（編集記号）の定義されている編集項目の場合は，編集が行われる。

・編集用PICTURE文字列は，データの種類や大きさを指定するとともに，編集方法を指定する。

・この文字列は，基本的なPICTURE文字(9, X)と編集用PICTURE文字の組合せである。

▶ 編集用PICTURE文字の種類

編集用 PICTURE文字	機　能
B	空白を挿入する文字位置を示す
/	斜線を挿入する文字位置を示す
0	ゼロを挿入する文字位置を示す
,	コンマを挿入する文字位置を示す
.	小数点を挿入する文字位置を示すとともに，位置合せの基準となる（実小数点）
¥	通貨記号の位置を示す
+ −	正負を示す文字（符号）の位置を示す
Z	数字データの左端から連続するゼロを空白に置き換える
*	数字データの左端から連続するゼロを星印に置き換える

▶ 編集方法の種類

編集を行うための一般的な方法は，挿入編集とゼロ抑制である。

編集の種類		編集用PICTURE文字	編集方法
挿入編集	単純挿入	B(空白) / 0 ,	指定された位置に挿入される
	特殊挿入	.（実小数点）	小数点の位置に挿入される
	固定挿入	¥ + −	指定された位置に挿入される
	浮動挿入	¥ + −	指定した範囲内で挿入位置が移動する
ゼロ抑制編集	空白によるゼロ抑制	Z	指定した範囲内で先行するゼロを空白に置き換える
	星印によるゼロ抑制	*	指定した範囲内で先行するゼロを"*"に置き換える

REDEFINES(再定義)句

REDEFINES 句は，コンピュータの同じ記憶領域に異なるデータ項目を定義するのに使用する。

▶ REDEFINES 句の構成

〈一般形式〉
レベル番号　データ名-1　<u>REDEFINES</u>　データ名-2

▶ 使用上の規則

①データ名-1とデータ名-2で示されるデータ項目のレベル番号は，同じでなければならない。

②ファイル節の 01 レベルの記述項に REDEFINES 句を指定することはできない。

③レベル番号 01 を除いて，データ名-1とデータ名-2で示されるデータ項目のけた数は，同じ大きさでなければならない。

④データ名-1のデータ項目の記述は，データ名-2のデータ記述の直後になければならない。ただし，同じ記憶領域を3回以上定義して利用する場合には，それらの記述を連続して行い，データ名-2として最初に定義したデータ項目の名前を指定する。

⑤データ名-2のデータ記述は，OCCURS 句を含んではならない。しかし，データ名-2は，データ記述に OCCURS 句が書いてある項目に従属してもよい。

	WORK-A		
	A 1	A 2	
		A 21	A 22
(PIC)	9 (5)	9 (7)	9 (3)

(再定義)

	WORK-B				
	B 1		B 2		B 3
	B 11	B 12	B 21	B 22	
(PIC)	X (2)	9 (3)	X (3)	9 (4)	X (3)

```
01  WORK-A.
    02  A1          PIC  9(5).
    02  A2.
        03  A21  PIC  9(7).
        03  A22  PIC  9(3).
01  WORK-B  REDEFINES  WORK-A.
    02  B1.
        03  B11  PIC  X(2).
        03  B12  PIC  9(3).
    02  B2.
        03  B21  PIC  X(3).
        03  B22  PIC  9(4).
    02  B3          PIC  X(3).
```

OCCURS(反復)句

OCCURS句は，同じ項目のデータ項目が繰り返されるとき，その記述項をいちいち指定する手間を省き，また，添字をつけるための情報を与える。OCCURS 句では，表(テーブル，配列)を定義する。

▶ OCCURS句の構成

〈一般形式〉

<u>OCCURS</u>　整数　TIMES

▶ 使用上の規則

① OCCURS 句は，レベル番号 01，77 で指定したデータ項目には使用できない。

② OCCURS 句のなか，またはそれにつづく項目の記述には VALUE 句は使用できない。

③ OCCURS 句で指定したデータ項目を手続き部で参照するときは，一意参照を行うために添字を用いる。

④添字はかっこを用い，OCCURS 句で指定された数の範囲内で指定する。

⑤かっこ内の添字が2つ以上あるときは，左のほうが階層の高い項目を示す。

⑥ OCCURS 句で定義される回数により，表の次元が決められる。

〔例 1〕1 次元の表

```
01 TABLE-1.
   02 A OCCURS 3 TIMES
        PIC X(4).
```

〔例 2〕2 次元の表

	TABLE-2											
	B				B				B			
	C	C	C	C	C	C	C	C	C	C	C	C
(PIC)	9	9	9	9	9	9	9	9	9	9	9	9

```
01 TABLE-2.
   02 B OCCURS 3 TIMES.
      03 C OCCURS 4 TIMES
           PIC 9(1).
```

定数表

表のなかのデータは，入力動作によって与えたり，処理の中間結果として与えることが多いが，初期値として与えることも多い。この初期値をもった表を定数表とよぶ。

定数表は，その表のなかから必要なものを探し出してデータ参照を行うことが多いので，いわゆる表引き（テーブルサーチ）といわれる処理で利用される。

▶ REDEFINES 句使用による定数表の定義

定数表は，初期値をもった表であることから，OCCURS 句と VALUE 句を用いて定義される。その方法は，最初に VALUE 句によって定数を定義し，その領域を再定義した後，OCCURS 句で表を定義する。

<table>
<tr><td></td><td colspan="12">TABLE-VALUE</td></tr>
<tr><td>(PIC)</td><td colspan="12">X (12)</td></tr>
<tr><td>(VALUE)</td><td>1</td><td>2</td><td>1</td><td>5</td><td>2</td><td>3</td><td>2</td><td>6</td><td>3</td><td>1</td><td>3</td><td>2</td></tr>
<tr><td></td><td colspan="12">(再定義)</td></tr>
<tr><td></td><td colspan="12">BU-TABLE</td></tr>
<tr><td></td><td colspan="2">BU-CODE</td><td colspan="2">BU-CODE</td><td colspan="2">BU-CODE</td><td colspan="2">BU-CODE</td><td colspan="2">BU-CODE</td><td colspan="2">BU-CODE</td></tr>
<tr><td>(PIC)</td><td colspan="2">X (2)</td><td colspan="2">X (2)</td><td colspan="2">X (2)</td><td colspan="2">X (2)</td><td colspan="2">X (2)</td><td colspan="2">X (2)</td></tr>
</table>

```
01  TABLE-VALUE  PIC  X(12)  VALUE  "121523263132".
01  BU-TABLE  REDEFINES  TABLE-VALUE.
    02  BU-CODE  OCCURS  6  TIMES  PIC  X(2).
```

▶ 集団項目による定数表の定義

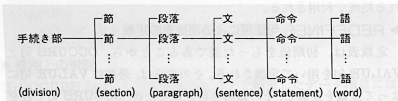
この方法は，集団項目で VALUE 句を用いて定数を定義し，下位レベルで OCCURS 句を用いて表を定義するものである。この方法では，文字定数がそのコンピュータで許される最大けたの範囲を超えると使用できないので注意しなければならない。

〔例〕
```
01  BU-TABLE  VALUE  "121523263132".
    02  BU-CODE  OCCURS  6  TIMES  PIC  X(2).
```

手続き部の構成と命令

手続き部は，環境部で指定されたファイルから，データ部で定義した記憶領域にデータを入力し，そのデータを演算加工して指定されたファイルに出力するための手続きを記述する。

▶ 手続き部の構成

手続き部の構成は，次のとおりである。

▶ 命令の種類

このハンドブックで扱う命令の種類は，次のとおりである。それぞれの命令は，アルファベット順に掲載した。

種　類	命　令
入出力命令	ACCEPT, CLOSE, DISPLAY, OPEN, READ, WRITE
算術命令	ADD, COMPUTE, DIVIDE, MULTIPLY, SUBTRACT
データ操作命令	MOVE, INSPECT
条件命令	IF
手続き分岐命令	GO TO, PERFORM
表操作機能	SEARCH, SET
整列機能	RELEASE, RETURN, SORT

〔補足〕条件命令には，IF 命令のほかに，条件指定（AT END, INVALID KEY, ON SIZE ERROR)のある入出力命令や算術命令も含まれる。

無条件命令とは，条件命令を除くすべての命令である。

ACCEPT 命令

▶ 一般形式

書き方 1

ACCEPT 一意名 [FROM 呼び名]

書き方 2

ACCEPT 一意名 FROM { DATE | DAY | TIME }

▶ 命令の機能と規則

①ACCEPT 命令は，少量のデータを指定されたデータ項目に格納する。

②書き方 1 は，作成者が定めた装置から一意名で示す領域にデータを読み込む。

③書き方 2 は，FROM 句で指定した命令実行時の日付または時刻を，一意名で示す領域に格納する。

- ・DATE——yymmdd 形式であり，yy は西暦の下 2 けた，mm は月，dd は日を表す。
- ・DAY——yyddd 形式であり，yy は西暦の下 2 けた，ddd は 1 月 1 日を 1 日目として数えた通日を表す。
- ・TIME——hhmmsstt 形式であり，hh は24時間表示，mm は分，ss は秒，tt は100分の 1 秒を表す。

④日付，時刻の格納は，MOVE 命令の転記規則に従う。

▶ 記述例

① ACCEPT SYAIN-CODE FROM CONSOLE.

制御卓 (CONSOLE) から社員コードを入力する。

② ACCEPT HIZUKE FROM DATE.

プログラムの処理日を格納する。

ADD命令

▶ 一般形式

書き方 1

```
ADD {一意名-1} … TO {一意名-2 [ROUNDED]} …
    {定数-1  }
                    [ON SIZE ERROR 無条件命令]
```

書き方 2

```
ADD {一意名-1} {一意名-2} … GIVING {一意名-n [ROUNDED]} …
    {定数-1  } {定数-2  }
                           [ON SIZE ERROR 無条件命令]
```

▶ 命令の機能と規則

①ADD命令は，2個以上の数字データ項目の和をとって，その結果を格納する。

②書き方1は，すべてのデータ項目が加算対象となり，TOの右にあるデータ項目に結果が格納される。

③書き方2は，GIVINGの左にあるデータ項目が加算対象となり，GIVINGの右にあるデータ項目に結果が格納される。

④一意名は数字基本項目，定数は数字定数でなければならない。

⑤ROUNDED句を指定すると，切捨て部分が四捨五入される。

⑥ON SIZE ERROR句を指定すると，けたあふれのときに無条件命令が実行される。

▶ 記述例

書き方1，2の各形式は，次のような加算となる。

①ADD A TO B. ……………………………B+A→B

②ADD A B TO C. ………………………C+(A+B)→C

③ADD A B GIVING C. …………………A+B→C

CLOSE 命令

▶ 一般形式

```
CLOSE  {ファイル名}…
```

▶ 命令の機能と規則

① CLOSE 命令は，ファイルの処理を終了させる。

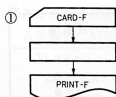
② CLOSE 命令を実行したファイルのレコード領域は，使用不可能となる。再び利用したいときは，OPEN 命令を実行しなければならない。

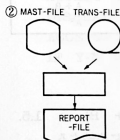
③ CLOSE 命令は，ラベルのあるファイルに対してラベルの処理も行う。

④ CLOSE 命令は，すでに開かれているファイルに対してだけ実行できる。

▶ 記述例

おのおののプロセスチャートで示されるファイルを閉じる場合の記述例は，次のとおりである。

①

```
CLOSE  CARD-F.
CLOSE  PRINT-F.
```

②

```
CLOSE  MAST-FILE.
CLOSE  TRANS-FILE.
CLOSE  REPORT-FILE.
```

または，

```
CLOSE  MAST-FILE
       TRANS-FILE
       REPORT-FILE.
```

COMPUTE 命令

▶ 一般形式

COMPUTE　{一意名 [ROUNDED]} … = 算術式
[ON SIZE ERROR 無条件命令]

▶ 命令の機能と規則

① COMPUTE 命令は，算術式の計算結果を格納する。

②右辺の算術式の値を左辺のデータ項目に格納する。

③算術式は次のいずれかである。

・数字基本項目の一意名か数字定数
・上記の一意名や定数を算術演算子でつないだもの
・算術式をかっこで囲んだもの

④ ROUNDED 句を指定すると，切捨て部分が四捨五入される。

⑤ ON SIZE ERROR 句を指定すると，けたあふれのときに無条件命令が実行される。

⑥算術演算子を次に示す。

意 味	演算子	COBOL 記述	代数式
加 算	+	A + B	A + B
減 算	−	A − B	A − B
乗 算	*	A * B	A × B
除 算	/	A / B	A ÷ B
べき乗	**	A ** 2	A^2

⑦算術式で表された演算の実行順序を次に示す。

かっこのなか→べき乗→乗除算→加減算

▶ 記述例

① COMPUTE C ROUNDED = (A + B) * 1.5.

AとBの和を5割増した値を四捨五入してCに格納する。

DISPLAY 命令

▶ 一般形式

DISPLAY {一意名 / 定数} … [UPON 呼び名]

▶ 命令の機能と規則

①DISPLAY 命令は，少量のデータを表示する。

②DISPLAY 命令は，一意名で示すデータ項目や定数の値を指定した順に，指定した装置に表示する。

③定数として表意定数を指定すると，表意定数の意味する文字の1けたを表示する。ただし，ALL 定数は指定できない。

▶ 記述例

① DISPLAY "END OF JOB".

"END OF JOB"というメッセージを表示する。

② DISPLAY "ケンスウ = " KENSU.

データの件数（KENSU）を表示する。

③あるクラスの成績表を出力するために，クラスコードを入力させる手順を，ACCEPT 命令と DISPLAY 命令で組み合わせる。

```
J1.
  DISPLAY "クラスコード゛ ノ キー イン".
  ACCEPT CLASS-CODE.
  DISPLAY "クラスコード゛ OK? Y OR N".
  ACCEPT Y-N.
  IF Y-N = "N" GO TO J1.
```

DIVIDE 命令

▶一般形式

書き方 1

```
DIVIDE {一意名-1} INTO {一意名-2 [ROUNDED]}…
       {定数-1 }
                  [ON SIZE ERROR 無条件命令]
```

書き方 2

```
DIVIDE {一意名-1} {INTO} {一意名-2}
       {定数-1 } {BY  } {定数-2 }
         GIVING {一意名-3 [ROUNDED]}…
              [ON SIZE ERROR 無条件命令]
```

書き方 3

```
DIVIDE {一意名-1} {INTO} {一意名-2}
       {定数-1 } {BY  } {定数-2 }
         GIVING 一意名-3 [ROUNDED]
         REMAINDER 一意名-4
              [ON SIZE ERROR 無条件命令]
```

▶命令の機能と規則

①DIVIDE 命令は，除算を行い，商と余りを格納する。

②一意名は数字基本項目，定数は数字定数である。

③ROUNDED 句を指定すると，切捨て部分が四捨五入される。

④REMAINDER 句を指定すると，剰余を一意名-4 に格納する。

⑤INTO と BY は，被除数と除数が逆になる。

▶記述例

書き方 1，2，3 の各形式は，次のような除算となる。

① DIVIDE A INTO B.……………………B÷A→B

② DIVIDE A BY B GIVING C. ……………A÷B→C

③ DIVIDE A INTO B GIVING C …………B÷A→C
REMAINDER D. 余り→D

GO TO命令

▶一般形式

書き方1
GO TO 手続き名-1
書き方2
GO TO {手続き名-1}…手続き名-*n* DEPENDING ON 一意名

▶命令の機能と規則

①GO TO命令は，手続き部のある部分からほかの部分に制御を移し，実行の順序を変更する。
②書き方1のGO TO命令は，実行順序を手続き名-1に変更する。
③書き方2のGO TO命令は，一意名の値が1，2……*n*であるとき，それぞれの手続き名-1，手続き名-2……手続き名-*n*に実行順序を変更する。一意名の値が正の整数以外の場合は，一般の実行順序における次の命令に制御が移る。
④書き方2において，一意名は整数項目でなければならない。

▶記述例

GO TO命令は，単独で用いる場合と，ほかの命令と組み合わせて用いる場合がある。

① GO TO J-1.　　手続きJ-1に制御を移す。
② IF A > 20 GO TO J-2.
　Aが20より大きい場合は，手続きJ-2に制御を移す。
③ READ CARD-F AT END GO TO J-3.　カードファイルを読み込み，もし終わりならば，手続きJ-3に制御を移す。
④ GO TO A-1 A-2 A-3 DEPENDING ON KUBUN.
区分コードが1のときA-1に，2のときA-2に，3のときA-3にそれぞれ制御を移す。区分コードが1，2，3以外のときは，次の命令が実行される。

IF 命令

▶一般形式

```
IF  条件 {命令-1          } {ELSE  命令-2          }
         {NEXT  SENTENCE} {ELSE  NEXT  SENTENCE}
```

▶ 命令の機能と規則

①IF 命令は，条件が真か偽かによって次に行う動作を定める。

②条件を評価して，

・真(YES)ならば，命令-1を実行するか，または次の文(NEXT SENTENCE) に制御を移し，ELSE 以降を無視する。

・偽(NO)ならば，ELSE の前までを無視し，命令-2を実行するか，または次の文に制御を移す。

③命令-1，命令-2のなかに IF 命令を書いてもよい。

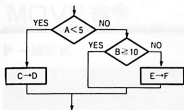
④ELSE NEXT SENTENCE の直後が終止符の場合には，この句を省略してよい。

▶ 条件の種類と書き方

条件には，比較条件，字類条件，正負条件，条件名条件などがある。なお，この条件は，PERFORM，SEARCH の各命令に共通であり，かっこが任意に使用できる。

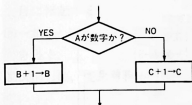
①比較条件：大小関係を調べるのに使う。なお，GREATER，LESS，EQUAL の使用も可能である。

〈一般形式〉

```
{一意名-1 } {IS [NOT] >} {一意名-2 }
{定数-1   } {IS [NOT] <} {定数-2   }
{算術式-1 } {IS [NOT] =} {算術式-2 }
```

```
IF  A < 5
    MOVE C TO D
ELSE
    IF  B NOT < 10
        NEXT SENTENCE
    ELSE
        MOVE E TO F.
```

②**字類条件**：数字であるか，英字であるかを調べるのに使う。

```
〈一般形式〉
一意名 IS [NOT] {NUMERIC   }
                {ALPHABETIC}
```

Aが数字か？
YES　NO
B+1→B　C+1→C

```
IF  A IS NUMERIC
    ADD 1 TO B
ELSE
    ADD 1 TO C.
```

③**正負条件**：算術式の代数値がゼロより大きいか，小さいかまたは等しいかを調べるのに使う。

```
〈一般形式〉
算術式 IS [NOT] {POSITIVE}
                {NEGATIVE}
                {ZERO    }
```

④**条件名条件**：データ項目（条件変数）の値が条件名に割り当てられている値に等しいかどうかを調べるのに使う。

・条件名は，データ部において，データ項目につづけて，レベル番号 88 を用いて記述する。

88 条件名 VALUE 定数-1 ……………………………特定の値
88 条件名 VALUE 定数-1 THRU 定数-2…………値の範囲
88 条件名 VALUE 定数-1 定数-2 …………………値の組合せ

これを，IF 条件名 ～ の形式で利用する。

⑤**複合条件**：各種の条件を論理演算子（AND, OR, NOT）を組み合わせて使う。

INSPECT 命令

▶ 一般形式

書き方 1

```
INSPECT 一意名-1 TALLYING

                   {{ALL     } {定数-1  }}
         一意名-2 FOR {{LEADING } {一意名-3}}
                   { CHARACTERS            }
```

書き方 2

```
INSPECT 一意名-1 REPLACING
  {CHARACTERS BY {定数-3  }                     }
  {              {一意名-5}                     }
  {{ALL    } {定数-2  }    {定数-3  }}
  {{LEADING} {        } BY {        }}
  {{FIRST  } {一意名-4}    {一意名-5}}
```

▶ 命令の機能と規則

① INSPECT 命令は，データ項目中の文字や文字列の出現回数を数えたり，それらをほかの文字や文字列で置き換えたりする。

② TALLYING 句は，一意名-1内の特定の文字や文字列を数えて，一意名-2の値に加える。

③ REPLACING 句は，一意名-1内の特定の文字や文字列を，定数-3や一意名-5で置き換える。

▶ 記述例

① INSPECT A TALLYING B FOR ALL "*".

データ項目 A 内の星印の出現回数をデータ項目 B に加える。

② INSPECT A REPLACING LEADING "△" BY "0".

データ項目 A 内の先行する空白をゼロに置き換える。

③ INSPECT A REPLACING ALL "△" BY "0".

データ項目 A 内のすべての空白をゼロに置き換える。

MOVE命令

▶一般形式

MOVE $\left\{\begin{array}{l}\text{一意名-1}\\\text{定数-1}\end{array}\right\}$ TO {一意名-2} …

▶命令の機能と規則

①MOVE命令は，データを転記規則に従って，いくつかのデータ項目に転記する。

②一意名-1および定数-1は送出し側を表し，一意名-2は受取り側を表す。

③送出し側，受取り側がともに基本項目の場合を，基本項目転記という。

- ・受取り側が数字項目または数字編集項目の場合，小数点の位置合せおよび必要なゼロづめが行われる。なお，数字編集項目の場合は，編集が行われる。
- ・受取り側が数字項目，数字編集項目以外の場合は，左側をそろえて転記され，余ったけたには空白づめが行われる。なお，英数字編集項目のときは，編集が行われる。

④基本項目転記以外は，左側をそろえて転記され，余ったけたには空白づめが行われる。ただし，編集は行われない。

▶記述例

① MOVE A TO B.

データ項目Aをデータ項目Bに転記する。

② MOVE ZERO TO C.

データ項目Cをゼロにする。

③ MOVE SPACE TO D.

データ項目Dを空白にする。

MULTIPLY 命令

▶ 一般形式

書き方 1

MULTIPLY {一意名-1 / 定数-1} BY {一意名-2 [ROUNDED]}…

[ON SIZE ERROR 無条件命令]

書き方 2

MULTIPLY {一意名-1 / 定数-1} BY {一意名-2 / 定数-2}

GIVING {一意名-3 [ROUNDED]}…

[ON SIZE ERROR 無条件命令]

▶ 命令の機能と規則

① MULTIPLY 命令は，データ項目どうしの積を計算し，結果を格納する。

②一意名は数字基本項目，定数は数字定数である。

③書き方 1 では，すべてのデータ項目が積の対象となり，BY の右にあるデータ項目に結果が格納される。

④書き方 2 では，GIVING の左にあるデータ項目が積の対象となり,GIVINGの右にあるデータ項目が,結果を格納する項目となる。

⑤ ROUNDED 句を指定すると，切捨て部分が四捨五入される。

⑥ ON SIZE ERROR 句を指定すると，けたあふれのときに無条件命令が実行される。

▶ 記述例

書き方 1，2 の各形式は，次のような乗算となる。

① MULTIPLY A BY B. ……………………………B×A → B

② MULTIPLY A BY B C. ……………………{B×A → B / C×A → C}

③ MULTIPLY A BY B GIVING C. ………B×A → C

④ MULTIPLY A BY B GIVING C D. {B×A → C / B×A → D}

OPEN命令

▶一般形式

$$\underline{\text{OPEN}} \left\{ \begin{array}{ll} \underline{\text{INPUT}} & \{\text{ファイル名-1}\}\cdots \\ \underline{\text{OUTPUT}} & \{\text{ファイル名-2}\}\cdots \end{array} \right\}\cdots$$

▶ 命令の機能と規則

① OPEN 命令は，ファイルの処理を行うための準備をする。

② OPEN 命令実行後は，そのファイルに対しレコード領域が使用可能になり，OPEN 命令以外の入出力命令が実行できる。

③ OPEN 命令は，ラベルのあるファイルに対してラベルの処理も行う。

④処理モード（入出力区分）を指定する。

- INPUT（入力）は，{ファイル名-1}…に対して，読込みだけが可能なことを意味する。
- OUTPUT（出力）は，{ファイル名-2}…に対して，書出しだけが可能なことを意味する。

▶ 記述例

おのおののプロセスチャートで示されるファイルを開く場合の記述例は，次のとおりである。

①
```
OPEN INPUT   CARD-F.
OPEN OUTPUT  PRINT-F.
```

②
```
OPEN  INPUT   MAST-FILE.
OPEN  INPUT   TRANS-FILE.
OPEN  OUTPUT  REPORT-FILE.
```
または，
```
OPEN  INPUT   MAST-FILE
              TRANS-FILE
      OUTPUT  REPORT-FILE.
```

PERFORM命令

▶ 一般形式

▶ 命令の機能と規則

①PERFORM 命令は，通常の実行順序から離れて，指定された手続きを実行し，再びもとの実行順序に戻る。

②一意名は，すべて数字基本項目(整数)でなければならない。

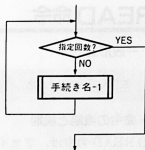
③定数は，すべて数字定数（整数）でなければならない。

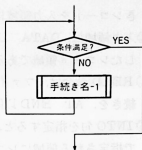
④条件は，IF 命令で述べた条件と同じである。

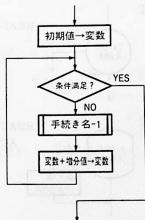
▶ 記述例

①書き方1は，指定された手続きを1回だけ実行する。

②書き方2は，一意名-1の初期値または整数-1で指定された回数だけ手続きを実行する。

```
PERFORM 手続き名-1
        回数 TIMES.
```

③書き方3は，条件-1を満足するまで，指定された手続きを反復実行する。

```
PERFORM 手続き名-1
        UNTIL 条件.
```

④書き方4は，PERFORM命令実行中に，一意名の値を規則的に変化させながら，指定された手続きを反復実行する。一意名-2が変数，一意名-3または定数-1が初期値，一意名-4または定数-2が増分値となる。

```
PERFORM 手続き名-1
   VARYING 変数
      FROM 初期値 BY 増分値
         UNTIL 条件.
```

READ 命令

▶一般形式

<u>READ</u>　ファイル名　［<u>INTO</u>　一意名］　AT　<u>END</u>　無条件命令

▶命令の機能と規則

①READ 命令は，ファイルからレコードを読み込む。

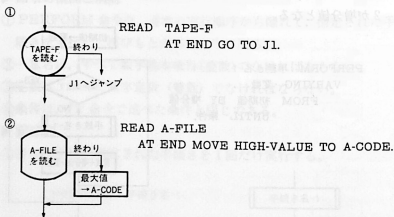
②READ 命令は，OPEN されているファイルから順次，処理すべきレコードを入力領域に読み込む。

③入力領域は，DATA　DIVISION の FILE　SECTION で定義したレコード領域である。

④READ 命令は，ファイル内データの終わりを検出した場合の手続きを，AT　END 以降に書き添える。

⑤INTO 句を指定すると，読み込まれたレコード領域から，一意名で指定される領域にレコード内容が転記される。

▶記述例

おのおののフローチャートで示されるコーディング例は，次のとおりである。

SUBTRACT命令

▶一般形式

書き方 1

```
SUBTRACT {一意名-1} … FROM {一意名-2 [ROUNDED]} …
         {定数-1  }
         [ON SIZE ERROR 無条件命令]
```

書き方 2

```
SUBTRACT {一意名-1} … FROM {一意名-2}
         {定数-1  }        {定数-2  }
         GIVING {一意名-n [ROUNDED]} …
         [ON SIZE ERROR 無条件命令]
```

▶命令の機能と規則

① SUBTRACT 命令は，指定されたデータ項目から，いくつかのデータ項目の和を引き，結果を格納する。

②一意名は数字基本項目，定数は数字定数でなければならない。

③ ROUNDED 句を指定すると，切捨て部分が四捨五入される。

④ ON SIZE ERROR 句を指定すると，けたあふれのときに無条件命令が実行される。

▶記述例

書き方 1，2 の各形式は，次のような減算となる。

① SUBTRACT A FROM B. ……………………… B－A → B

② SUBTRACT A B FROM C. ……………… C－(A＋B)→ C

③ SUBTRACT A FROM B C. ………………… { B－A → B, C－A → C }

④ SUBTRACT A FROM B GIVING C. ……B－A → C

WRITE命令

▶一般形式

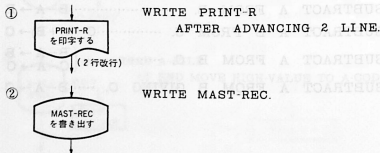
▶命令の機能と規則

①WRITE 命令は，OPEN されている出力ファイルに，処理した結果を書き出す。

②FROM 句を指定すると，一意名-1の領域からレコード名で指定された領域にデータを転記した後に書き出される。

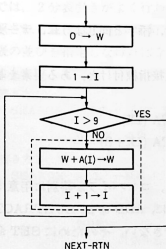
③プリントファイルに対して，改行や改ページを指定するためにADVANCING 句がある。

- 一意名-2または整数は改行数を示す。
- PAGE を指定すると改ページが行われる。
- AFTER と BEFORE は，レコードの書出しを，行送り後に行うか行送り前に行うかの指定である。なお，この AFTER または BEFORE 以降を省略すると， AFTER ADVANCING 1 LINE とみなされる。

▶記述例

添字による表操作

OCCURS 句で定義された表のなかの特定のデータ項目を一意に参照するために，添字（subscript）が用いられる。すなわち，表の各要素は，その出現番号を添字として用いることにより一意となり，表操作が実現される。

添字には，整数またはデータ名を書く。特に，添字にデータ名を書くことによって，そのデータ領域の値で出現番号を指定できるので，反復処理が可能となる。

次の例は，表中の 9 個のデータ項目の総和を求める処理である。例 1 は，IF 命令を用いたコーディング例であり，例 2 は，PERFORM 命令を用いたコーディング例である。

指標による表操作

OCCURS 句で表を定義するときに，指標名を定義すると，指標名による表操作を行うことができる。

▶ INDEXED BY 句

OCCURS 句につづいて，INDEXED　BY 句を指定することにより，指標に名前（指標名）をつけることができる。これによって，指標とよばれる領域が自動的に確保される。

〈一般形式〉

書き方 1

```
    レベル番号  データ名-1  OCCURS  整数  TIMES
               [INDEXED  BY  {指標名-1}…]
```

書き方 2

```
    レベル番号  データ名-1  OCCURS  整数  TIMES
               [{ASCENDING }  KEY  IS  {データ名-2}…]
               [{DESCENDING}                         ]
               [INDEXED  BY  {指標名-1}…]
```

▶ 指標付け

指標名によって表操作を行うときは，添字と同じように，データ名に指標をつけて一意に参照する。

指標付けには，添字と同じ方法の直接指標付けと，ある要素を基準とした相対指標付けがある。

〔例〕　直接指標付け：TABLE-DATA (IX)

　　　　相対指標付け：TABLE-DATA (IX+3)

▶ SET 命令

指標は通常のデータ項目とは異なり，コンパイラが特別に用意する領域なので，指標の値を設定するのに，ADD 命令，SUBTRACT 命令および MOVE 命令などは使用できない。そのために SET 命令が用意されている。

〈一般形式〉

書き方 1

```
SET {(指標名-1)…} TO {指標名-2
    {(一意名-1)…}    {一意名-2
                     {整数-1
```

書き方 2

```
SET (指標名-3)… {UP BY  } {一意名-3}
                {DOWN BY} {整数-2  }
```

▶ SEARCH 命令

表のなかから，ある条件に合った要素を探し出す操作を表引きという。COBOL では，SEARCH 命令により，指標を用いて表引きを行うことができる。

SEARCH 命令は，指定した条件を満足する表の要素を探し出し，対応する指標がその表の要素を指すようにする。

書き方 1 は，逐次表引きを行うときに使用される。このとき SEARCH 命令は，逐次，指標の値を増加させて表の要素を検査する。

書き方 2 は，非逐次表引きを行うときに使用される。非逐次表引きでは，2 分表引きがよく行われる方法である。なお，書き方 2 で参照される表は，OCCURS 句で定義するとき，KEY IS 句を用いて表の並びを指定しなければならない。

〈一般形式〉

書き方 1

```
SEARCH 一意名-1 [AT END 無条件命令-1]
        WHEN 条件 {無条件命令-2   }
                  {NEXT SENTENCE}
```

書き方 2

```
SEARCH ALL 一意名-1 [AT END 無条件命令-1]
                                {一意名-2}
        WHEN {データ名 IS  =    {定  数  }}
             {                  {算術式  }}
             {条件名                      }
```

整列機能

事務計算の分野では，整列処理はきわめて重要な役割を果たす。整列には，主記憶のなかで行う内部整列と，補助記憶を使って行う外部整列がある。COBOL では，外部整列を行うために，整列機能が用意されている。

COBOL の整列機能は，指定したキーに従ってファイルのレコードの順序をそろえたり，そろえる前やそろえた後に特別な処理（入力手続き，出力手続き）を行うことができる。これらの処理は，SORT 命令によって実現する。

▶整列機能の構成

整列機能の全体の構成は次のとおりである。

```
          ⌇
ENVIRONMENT        DIVISION.
          ⌇
INPUT-OUTPUT       SECTION.
FILE-CONTROL.
          ⌇
┌──────────────────────────────────────────────────┐
│    SELECT  整列用ファイル名  ASSIGN  TO  作成者語.  │
└──────────────────────────────────────────────────┘
          ⌇
DATA       DIVISION.
FILE       SECTION.
          ⌇
┌──────────────────────────────────────────────────┐
│SD    整列用ファイル名.                              │
│01    [レコード記述]                                 │
└──────────────────────────────────────────────────┘
          ⌇
PROCEDURE       DIVISION.
          ⌇
┌──────────────────────────────────────────────────┐
│    SORT  整列用ファイル名                           │
│              ⋮                                      │
└──────────────────────────────────────────────────┘
```

▶整列用ファイルの定義

整列機能では，補助記憶装置を使用するために，整列用ファイルを定義しなければならない。整列用ファイルは，環境部で定義したファイルについて，データ部において，レベル指示語SDとファイル名を書き，その後に，レコード記述項を書く。

なお，整列用ファイルには，OPEN，READ，WRITE，CLOSEなどの通常の入出力命令は使用できない。

```
FILE SECTION.
SD 整列用ファイル名.
 レコード記述項
```

▶ SORT命令の機能

整列を行うときはSORT命令を使用する。SORT命令は，整列用ファイルに引き渡されたレコードを，指定されたキーに従って並べ替える。

整列には，単純な整列と，入力手続き・出力手続きを含む整列とがある。単純な整列は，入力ファイルの全レコードを整列後，出力ファイルに書き出す。これに対して，入力手続き・出力手続きを含む整列では，入力ファイルから整列用ファイルへレコードを引き渡すさい，あるいは，整列後に整列用ファイルからレコードを引き取るさいに，データを加工する処理ができる。

整列前にデータを加工するときは，INPUT PROCEDURE 句を用いて入力手続きを指定し，整列後にデータの加工を行うときは，OUTPUT PROCEDURE 句を用いて出力手続きを指定する。

入力手続きを行わないときは，USING 句で入力ファイルを指定し，出力手続きを行わないときは，GIVING 句で出力ファイルを指定すればよい。

整列の基準となるキーは，ASCENDING（昇順）句または DESCENDING（降順）句で指定する。指定した順番に，第1キー，第2キーというように，優先順位が与えられる。

▶ RELEASE 命令と RETURN 命令

入力手続きにおいて，加工したレコードを整列用ファイルに引き渡すときは，RELEASE 命令を使用する。また，出力手続きにおいて，整列済みのデータを引き取るときは，RETURN 命令を使用する。

RELEASE 命令中のレコード名は，整列用ファイルのレコード名であり，RETURN 命令中のファイル名は，整列用ファイルのファイル名である。この2つの命令は，通常のファイルで使用する WRITE 命令，READ 命令とほぼ同様な機能を果たす。ただし，前述のように，OPEN 命令，CLOSE 命令が不要なので注意しなければならない。

〈RELEASE 命令の形式〉

RELEASE　レコード名　[FROM　一意名]

〈RETURN 命令の形式〉

RETURN　ファイル名　RECORD　[INTO　一意名]
　　　　　　　　　　　AT　END　無条件命令

予約語表(JIS COBOL　C 6205-1980)

ACCEPT
ACCESS
ADD
ADVANCING
AFTER
ALL
ALPHABETIC
ALSO
ALTER
ALTERNATE
AND
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR

BEFORE
BLANK
BLOCK
BOTTOM
BY

CALL
CANCEL
CD
CF
CH
CHARACTER
CHARACTERS
CLOCK-UNITS
CLOSE
COBOL
CODE
CODE-SET
COLLATING
COLUMN
COMMA
COMMUNICATION
COMP
COMPUTATIONAL
COMPUTE
CONFIGURATION
CONTAINS
CONTROL
CONTROLS
COPY
CORR
CORRESPONDING
COUNT
CURRENCY

DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DE
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DEBUGGING
DECIMAL-POINT
DECLARATIVES
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DESTINATION
DETAIL
DISABLE
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC

EGI
ELSE
EMI
ENABLE
END
END-OF-PAGE
ENTER
ENVIRONMENT
EOP
EQUAL
ERROR
ESI
EVERY
EXCEPTION
EXIT
EXTEND

FD
FILE
FILE-CONTROL
FILLER
FINAL
FIRST
FOOTING
FOR
FROM

GENERATE
GIVING

GO
GREATER
GROUP

HEADING
HIGH-VALUE
HIGH-VALUES

I-O
I-O-CONTROL
IDENTIFICATION
IF
IN
INDEX
INDEXED
INDICATE
INITIAL
INITIATE
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
IS

JUST
JUSTIFIED

KEY

LABEL
LAST
LEADING
LEFT
LENGTH
LESS
LIMIT
LIMITS
LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINKAGE
LOCK
LOW-VALUE
LOW-VALUES

MEMORY
MERGE
MESSAGE
MODE
MODULES
MOVE
MULTIPLE
MULTIPLY

NATIVE
NEGATIVE
NEXT
NO
NOT
NUMBER
NUMERIC

OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
ORGANIZATION
OUTPUT
OVERFLOW

PAGE
PAGE-COUNTER
PERFORM
PF
PH
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PRINTING
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM-ID

QUEUE
QUOTE
QUOTES

RANDOM
RD
READ
RECEIVE
RECORD
RECORDS
REDEFINES
REEL
REFERENCES
RELATIVE
RELEASE
REMAINDER
REMOVAL

RENAMES
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RETURN
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN

SAME
SD
SEARCH
SECTION
SECURITY
SEGMENT
SEGMENT-LIMIT
SELECT
SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SIGN
SIZE
SORT
SORT-MERGE
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
START
STATUS
STOP
STRING
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUBTRACT
SUM
SUPPRESS
SYMBOLIC
SYNC
SYNCHRONIZED

TABLE
TALLYING
TAPE
TERMINAL
TERMINATE
TEXT
THAN
THROUGH
THRU
TIME
TIMES
TO
TOP
TRAILING
TYPE

UNIT
UNSTRING
UNTIL
UP
UPON
USAGE
USE
USING

VALUE
VALUES
VARYING

WHEN
WITH
WORDS
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZEROS

+
−
*
/
**
>
<
=

和文索引（50音順）

あーお

かーこ

さーそ

たーと

なーの

はーほ

まーも

やーよ

らーろ

欧文索引

月刊
情報処理試験
[6月号別冊付録]

CASL
FORTRAN
COBOL

▽

基本文法ハンドブック